Predictable Data Transport: A Delay and Energy Perspective

Dissertation zur Erlangung des Grades des
Doktors der Ingenierwissenschaften (Dr.-Ing.)
der Fakultät für Mathematik und Informatik
der Universität des Saarlandes

vorgelegt von

Pablo Gil Pereira

Saarbrücken,
2024

Tag des Kolloquiums
24. Juni 2024
Dekan der Fakultät
Prof. Dr. Roland Speicher

Prüfingsausschuss

Vorsitz
Prof. Dr. Isabel Valera
Berichterstatter
Prof. Dr.-Ing. Thorsten Herfet
Prof. Dr.-Ing. Timo Hönig
Prof. Dr.-Ing. Pietro Manzoni
Beisitzer
Dr.-Ing. Andreas Schmidt

Short Abstract

Cyber-physical systems extend the digital revolution to almost every aspect of our lives by bridging the gap between the digital and physical worlds. These systems demand unprecedented timeliness and reliability guarantees that the current operating and network systems do not provide. Transport layer protocols are the direct communication interface for the application layer and, hence, are key to providing end-to-end guarantees to the application. This thesis addresses how transport layer protocols should be designed to support cyber-physical systems. A clear candidate is the Predictably Reliable Real-time Transport (PRRT) protocol, which provides the application with a predictably reliable service within the specified time budget. This thesis makes original contributions to PRRT's error control function, which decides when and how much redundancy must be transmitted to meet the reliability and delay requirements of the application. The main contributions of this thesis are threefold: i) the SHARQ algorithm, which obtains the optimal error control configuration meeting the application constraints, and has been optimized to achieve predictably quick reactions to channel changes, ii) the DeepSHARQ algorithm, which leverages neural networks and a novel output regularization method to bring this predictability to resource-constrained devices, and iii) a systematic analysis of binary codes as an energy-efficient alternative for error coding at the transport layer, questioning the long-held belief that Vandermonde codes are a more suitable alternative due to their better error correction capabilities.

Kurze Zusammenfassung

Cyber-physikalische Systeme erweitern die digitale Revolution auf nahezu jeden Aspekt unseres Lebens, indem sie die Kluft zwischen der digitalen und physikalischen Welt überbrücken. Diese Systeme erfordern beispiellose zeitliche und beispiellose Garantien bzgl. Zeit-Verbrauch und Fehlerraten, die das aktuelle Betriebssystem und das Netzwerk nicht bieten. Transport-Protokolle sind die direkte Kommunikationsschnittstelle für die Anwendungsebene und sind daher entscheidend für die Bereitstellung von Ende-zu-Ende-Garantien für die Anwendungen. Diese Dissertation beschäftigt sich damit, wie Transport-Protokolle gestaltet sein sollten, um Cyber-physikalische Systeme zu unterstützen. Ein ausgezeichneter Kandidat ist das PRRT-Protokoll, das der Anwendung einen vorhersagbar zuverlässigen Dienst innerhalb des festgelegten Zeitbudgets bietet. Diese Dissertation leistet originelle Beiträge zur Fehlerkontrollfunktion von PRRT, die entscheidet, wie viel, und wann Redundanz übertragen werden muss, um die Zuverlässigkeits- und Verzögerungsanforderungen der Anwendung zu erfüllen. Die Hauptbeiträge dieser Dissertation sind in drei Teile auf: i) Der SHARQ-Algorithmus, der die optimale Konfiguration der Fehlerkontrolle gemäß den Einschränkungen ermittelt, wurde optimiert um vorhersehbare schnelle Reaktionen auf Kanaländerungen zu erzielen, ii) Der DeepSHARQ-Algorithmus, welcher Neuronale Netzwerke nutzt, und eine neuartige Ausgabe-Regularisierungsmethode, um diese Vorhersagbarkeit auf ressourcenbeschränkte Geräte zu übertragen, iii) eine systematische Analyse von Binärcodes als energieeffiziente

Alternative für die Fehlercodierung auf der Transport-Schicht, wobei die langjährige Überzeugung hinterfragt wird, ob Vandermonde-Codes aufgrund ihrer besseren Fehlerkorrekturfähigkeiten wirklich die bessere Alternative sind.

Abstract

Cyber-physical systems bridge the gap between the digital and physical worlds by closing control loops over embedded computers and networks. Given their positive impact on improved efficiency and performance, they are slowly permeating more and more aspects of our lives (i.e., connected vehicles, smart industry, distributed energy generation, telemedicine, etc.), extending the digital revolution beyond purely information-intensive processes to safety-critical applications. For digital systems to be deployed in safety-critical environments, their timeliness and reliability guarantees must be brought to unprecedented levels to ensure they do not put human lives at risk.

Current computing and networking systems do not offer abstractions to applications that guarantee predictable timing and reliability. Therefore, these abstraction layers must be redesigned to provide a dependable substrate for the deployment of cyber-physical systems. Transport layer protocols are the direct communication interface for the application layer and, hence, are key to providing end-to-end guarantees to the application. This thesis addresses how transport layer protocols should be designed to provide a predictable communication channel to the application, even on resource-constrained devices, the natural components of cyber-physical systems.

Traditional transport protocols either fail to provide delay guarantees (TCP) and reliability guarantees (UDP) or operate far from the channel capacity (RTP). The Predictably Reliable Real-time Transport (PRRT) protocol developed at the Telecommunications Lab at Saarland Informatics Campus addresses these issues by providing a predictably reliable service within a delay budget. PRRT implements a novel error control mechanism that finds the balance between proactive and reactive redundancy that best approaches the channel capacity—i.e., transmits the minimum redundancy. This thesis makes original contributions to the components of this error control mechanism with the highest computational complexity, namely the optimization algorithm that searches for the optimal configuration meeting the application requirements and the coding of parity packets to correct losses.

First, a computational complexity analysis of the optimization algorithm is presented together with the SHARQ algorithm. SHARQ reduces the number of CPU cycles required to find the optimal configuration meeting the application constraints for the current channel conditions. SHARQ improves upon previous search algorithms with a more efficient solution space exploration and a fast parity packet scheduling among the repair cycles. As a result of its low inference times, SHARQ enables PRRT to pro-

vide predictable communication channels on high-end devices due to the fast reaction to channel changes. However, SHARQ's computational complexity is still too high for PRRT to be deployed on resource-constrained embedded devices.

DeepSHARQ leverages neural networks to reduce the inference time further, thereby sacrificing optimality while improving the runtime predictability on low-end devices. An advantage of neural networks is their constant computational complexity, so their inference time solely depends on the network architecture and the platform on which it is executed. DeepSHARQ's neural networks are trained with a novel output regularization mechanism that reduces the learning complexity of the problem to obtain smaller neural networks than if traditional learning methods were employed. Output regularization allows DeepSHARQ to learn a set of configurations that maintain the transmitted redundancy within configurable margins from the minimum redundancy. In other words, the deviation from the optimum is predictable and configurable. The presented results show that DeepSHARQ brings predictably quick reactions to channel changes to resource-constrained devices for the first time.

Finally, the thesis also addresses energy aspects of the parity packet coding mechanisms in PRRT. The complexity dilemma has driven the development of error coding in the last decades such that deviations from the channel capacity were allowed to reduce computational complexity with algorithms that do not implicitly invert the generator matrix. As throughput—and hence block length—increases with new physical layer technologies, the redundancy overhead becomes negligible. This thesis proves that the complexity dilemma is entirely different in the transport layer, where the complexity is no longer dominated by the matrix inversion but by the matrix-vector multiplication instead, which must be iterated throughout complete packets. In light of this result, and to the best of our knowledge, we provide the first systematic analysis of the suitability of binary codes as an energy-efficient alternative to the optimal Maximum Distance Separable (MDS) codes in the transport layer. Despite the significant redundancy increase binary codes introduce in the transport layer, this thesis shows that binary codes demand less energy in a broad range of different network conditions and application requirements.

Acknowledgments

First and foremost I would like to thank Thorsten Herfet for giving me the opportunity to pursue a Ph.D., and for his guidance and advice in the last 6 years that helped keep my curious and wandering mind on track. I will certainly miss the work environment at the Telecommunications Lab, where I was allowed to pursue my own ideas and learn from my mistakes. I would also like to thank Timo Hönig for acting as a reviewer of this thesis and for his insights and pieces of advice throughout the LARN and e.LARN project. I am also grateful to Pietro Manzoni not only for acting as a reviewer for this thesis but also for organizing several editions of the CCNC conference, a venue I loved attending for its wonderful community.

It has been a pleasure to work at the Telecommunications Lab with some of the most talented people I have ever met. I am sincerely grateful to Andreas for accepting me as a master's student more than 7 years ago, whose eye for detail, disposition toward open discussions, and ability to create an excellent team environment greatly influenced my decision to pursue a Ph.D. I also want to thank Tobias for guiding my first steps at the chair and always being ready to help despite the ton of work that sat at his desk. Thanks to Kai for our insightful discussions and the fun we had either optimizing algorithms or climbing routes in KBA. Thanks to Marlene for your inspiring work ethic and positivity, and Robin for being an excellent office mate, who was always willing to discuss scientific topics outside of our research areas. Thanks to my "hermano" Misha for having popped up in my last months at the chair with his sharp sense of humor and Russian sweets. I have also been lucky to work with many students, namely Amina, Sven, Moritz, Nikolai, and Julius, from whom I probably learned more than they could ever learn from my advice. I am also thankful to Zakaria for all his support and advice, as well as our complaints about how poorly Bayern and Sevilla played over some tasty food at Philo Cafe. Lastly, I want to thank Diane for always being ready to help with any non-scientific process, and making them always run smoothly.

I am also thankful to Pascal for cheering me up with his extremely bad dad jokes, and Nicolas for being an excellent gym partner, which helped me keep both my physical and mental health through the arduous process of writing this document. I would also like to thank Edoardo and Herbert for sticking together since our first month in Saarbrücken more than 7 years ago. They made my first year in Germany one of the most exciting of my life, full of thrilling trips and funny anecdotes. I am grateful to Edoardo for his unique sense of humor, the never-ending recommendations of good blues music, and his

crazy bike trip from Vernante to Huelva. Thanks to Herbert for being the funniest and most unpredictable guide in Prague one could have. I am also grateful to Elena, Vale, and Hossein for our remote book club and Among Us sessions, which made being stuck at home for months significantly less burdensome.

I would also like to mention Eloy, Javi, and Andrés, who make it feel as if we all had never left Huelva after all these years. Although our busy lives made them more sporadic, I still love our long chats, and their support was always felt despite the distance. I want to thank Eloy for being an excellent friend who never gives up on others and is one of the most determined and resilient people I know. Thanks to Javi for still being there after more than 30 years. Thanks to Andrés for stimulating my critical thinking with many insightful conversations. I want to thank Paula and Marce as well for their warm welcome back home and their amazing amaretto sour, both of which made writing the last chapters of this thesis a more enjoyable experience.

I would also like to thank my siblings Marta and Jorge for their support despite the strange looks they give me every time I try to explain what I do. Thanks to my parents Macarena and Emilio for their strenuous work in the hardest circumstances, which fundamentally shaped who I am. I am also grateful to have two lovely grandparents, Enrique and Mari Carmen, who taught me how to enjoy life since I was little. Finally, I want to thank Kay for still loving me despite having to cope with a continuous stream of dummy jokes and for her endless patience. I love you too and I am looking forward to our next chapter.

Contents

1 Introduction **1**
- 1.1 Problem Statement . . . 2
- 1.2 Research Question . . . 3
- 1.3 Contributions . . . 4
- 1.4 Outline . . . 6

2 Distributed Cyber-Physical Systems **9**
- 2.1 Cyber-Physical Systems . . . 9
- 2.2 Energy, Power, and Sustainability . . . 12
- 2.3 The Centrality of the Transport Layer . . . 15
- 2.4 Conclusion . . . 23

3 Predictably Reliable, Real-Time Transport **25**
- 3.1 Design Principles . . . 25
- 3.2 Cross-Layer Pacing . . . 28
- 3.3 Adaptive HARQ . . . 30
- 3.4 Packet Format . . . 35
- 3.5 Optimal Hybrid Erasure Coding . . . 36
- 3.6 The need for HARQ . . . 40
- 3.7 Discussion . . . 42

4 Search Algorithms for Adaptive HARQ **45**
- 4.1 Predictable Communication Channel . . . 47
- 4.2 Full Search . . . 50
- 4.3 SHARQ: Scheduled HARQ . . . 52
- 4.4 Search Comparison . . . 59
- 4.5 Discussion . . . 66
- 4.6 Related Work . . . 68
- 4.7 Conclusion . . . 69

5 Deep Learning for Adaptive HARQ **71**
- 5.1 Learning How to Balance . . . 71
- 5.2 DeepSHARQ: Hybrid Error Coding using Deep Learning . . . 76

5.3 Model Training . . . 82
5.4 Performance Analysis . . . 90
5.5 Discussion . . . 94
5.6 Related Work . . . 97
5.7 Conclusion . . . 99

6 Energy-Aware Adaptive HARQ 101
6.1 The Complexity Dilemma (Revisited) . . . 101
6.2 Binary Erasure Codes . . . 105
6.3 Theoretical Analysis . . . 112
6.4 Energy-Aware HARQ . . . 118
6.5 Practical Analysis . . . 124
6.6 Discussion . . . 127
6.7 Related Work . . . 129
6.8 Conclusion . . . 131

7 Conclusion 133
7.1 Summary . . . 133
7.2 Future Work . . . 135

A Computational Complexity Packet Loss Rate 137

B Computational Complexity Redundancy Information 139

Own Publications 141

Bibliography 143

List of Figures

1.1 Summary of Contributions 6

2.1 Error control schemes 18
2.2 Systematic coding 19

3.1 PRRT Architecture 28
3.2 Paced vs. unpaced communication 29
3.3 X-Pace architecture 29
3.4 Loss detection algorithm 30
3.5 Loss detection delay 31
3.6 Matrix-vector multiplication 34
3.7 PRRT General Header 35
3.8 PRRT Data Header 35
3.9 PRRT Parity Header 36
3.10 PRRT Feedback Header 36
3.11 PRRT NAK Header 37
3.12 HARQ delay budget 38
3.13 Redundancy information increase 43

4.1 Solution space non-linearities 46
4.2 Performance predictability 49
4.3 Number of restricted integer compositions 52
4.4 $PLR_{HARQ}(k,p)$, $p_{opt}(k)$, and $RI(k,N_C,N_P)$ functions 54
4.5 Decreasing cycle probability 55
4.6 Graph search example 57
4.7 Number of evaluated configurations by search algorithm 62
4.8 CDF number of configurations 63
4.9 SHARQ inference time 64

5.1 Artificial neuron 72
5.2 Search algorithms information flow 80
5.3 DeepSHARQ's neural network architecture 81
5.4 Impact of learning rate and schedule on model performance 85
5.5 RI increase by neural network architecture and Δ 87

5.6 Neural network inference time 89
5.7 Model conversion pipeline. 90
5.8 DeepSHARQ inference time 92
5.9 Optimal vs. Simple Schedule Inference Time 94
5.10 Optimal vs. Simple Schedule Data Rate Increase 95

6.1 Computational complexity of matrix operations 104
6.2 Cumulative distribution function of the optimal k and p 105
6.3 Tanner graph representation of a binary generator matrix 106
6.4 Decoding failure probability of a random fountain code 107
6.5 Probability of losing the complete codeword in multicast 110
6.6 Encoding complexity of MDS and random fountain codes 113
6.7 Number of excess packets in random fountain codes vs. LT codes 114
6.8 Erasure recovery probability in MDS, random fountain, and polar codes . 116
6.9 Number of parity packets in MDS, random fountain, and polar codes . . . 117
6.10 Energy measurement hardware setup 120
6.11 Power draw borders detection 121
6.12 Coding energy demand in MDS and binary codes 123
6.13 Energy demand of packet transmission and reception 123
6.14 Energy-aware performance comparison 125
6.15 MDS and binary codes' energy demand by application requirements . . . 126

List of Tables

3.1 Comparison of transport protocols 27
3.2 Dataset parameters 41
3.3 Valid configurations by error control scheme 42

4.1 Run-time complexity of the basic performance metrics 61
4.2 SHARQ predictability requirements 66
4.3 SHARQ ϵ-fulfilling analysis 66

5.1 Hyperparameter analysis of regularization and epochs 86
5.2 DeepSHARQ neural network performance $\Delta = 0.1$ 88
5.3 DeepSHARQ$_{opt}$ neural network performance 89
5.4 DeepSHARQ$_{full}$ neural network performance 90
5.5 DeepSHARQ ϵ-fulfilling analysis 93

6.1 Binary codes and their adoption in cellular networks. 103
6.2 Linear functions to obtain the energy demand in milliJules 122
6.3 Linear functions to obtain the delay in milliseconds. 124
6.4 Energy demand comparison between MDS and binary codes 126

Acronyms

3G Third-generation broadband cellular networks. 19, 103, 130

4G Fourth-generation broadband cellular networks. 41, 83, 103

5G Fifth-generation broadband cellular networks. 4, 19, 25, 41, 83, 103, 110, 135

ABR Adaptive Bit Rate. 39, 98

ACK Acknowledgment. 17, 20, 21, 30–32, 36, 39, 68, 69, 79, 98

AI Artificial Intelligence. 73

API Application Programming Interface. 26, 27, 29, 44

ARM Advanced RISC Machines. 131

ARQ Automatic Repeat reQuest. 3, 4, 17, 21, 22, 27, 30–32, 38–40, 42, 43, 50, 79, 95, 133

BBR Bottleneck Bandwidth and Round-trip propagation time. 20, 29, 33, 98

BEC Binary Erasure Channel. 110, 111, 117

CCA Congestion Control Algorithm. 16, 20

CDF Cumulative Distribution Function. 42, 43, 63, 64, 89, 91, 92, 105, 115, 116, 125

CNN Convolutional Neural Network. 74, 75, 97

CPN Cyber-Physical Networking. 1, 4, 10

CPS Cyber-Physical System. 1–7, 9–15, 19, 23, 25–27, 31, 41–43, 47, 48, 59, 61, 66, 67, 70, 74, 76, 101, 127, 133–135

CPU Central Processing Unit. v, 2, 12, 14, 44, 61, 67, 74, 85, 91, 94, 95, 105, 118, 124, 127, 130

DASH Dynamic Adaptive Streaming over HTTP. 20, 22, 39, 98

DCCP Datagram Congestion Control Protocol. 22, 25

DCTPC Data Center TCP. 20

DeepSHARQ Deep-learned, Scheduled HARQ. iii, vi, 5–7, 76–83, 86, 88–97, 99, 124, 128, 135, 136

DetNet Deterministic Networking. 12, 25

DL Deep Learning. 71, 72, 74, 97–99, 128, 134

DNN Deep Neural Network. 71–75, 97–99

DTLS Datagram Transport Layer Security. 27

DUT Device Under Test. 119, 120

E2E End-To-End. 29

ECN Explicit Congestion Notification. 20, 68

FCT Flow Completion Time. 19, 20, 69

FEC Forward Error Coding. 4, 17, 20–22, 27, 30, 32, 38–40, 42, 43, 52, 68, 69, 79, 97, 98, 133

GCC Google Congestion Control. 22

GPU Graphics Processing Unit. 74, 127, 129, 130

GreenAI Green Artificial Intelligence. 73, 74

HARQ Hybrid Automatic Repeat reQuest. 4–7, 17–19, 25, 27, 30, 33, 36–44, 48, 50, 63, 69, 71, 75–77, 89, 94–96, 99, 103, 118, 119, 122, 124, 126, 128, 129, 131, 133–137

HEC Hybrid Error Coding. 4

HPCC High Precision Congestion Control. 20

HTTP Hypertext Transfer Protocol. 3, 15

I2C Inter-Integrated Circuit. 9, 119

IETF Internet Engineering Task Force. 48

IoT Internet of Things. 2, 11, 14, 16, 101, 117, 130, 131

IP Internet Protocol. 1, 3, 15, 19, 34, 41, 69, 83, 103, 129, 130, 135

IPCC Intergovernmental Panel on Climate Change. 2, 12, 13

LDPC Low-Density Parity Check. 21, 103

LLS Linear Least Squares. 121, 122

LSTM Long Short-Term Memory. 44, 97, 98, 135

LT Luby Transform. 21, 106, 108, 109, 112–114, 130, 131

LTP Loss-tolerant Transmission Protocol. 22

MAE Mean Absolute Error. 71

MDS Maximum Distance Separable. vi, 6, 32, 37, 39, 44, 57, 80, 101–104, 106–109, 112–115, 117, 119–128, 130, 131, 134, 137

ML Machine Learning. 44

MP Message Passing. 105, 108, 120

MPTCP Multipath TCP. 69

MSE Mean Squared Error. 71

MTU Maximum Transmission Unit. 19, 21, 41, 83, 103

NAK Negative Acknowledgment. 31, 32, 35–37, 130

NAS Neural Architecture Search. 75

NN Neural Network. 86, 91, 93

NTP Network Time Protocol. 44

ONNX Open Neural Network Exchange. 91

OS Operating System. 3, 15

PC Personal Computer. 59, 61, 63–66, 70, 71, 86, 89–91, 93, 94, 119, 121, 124, 127, 134

PCC Performance-oriented Congestion Control. 20

PLR Packet Loss Rate. 40, 51, 53, 54, 61, 62, 80, 117, 137, 138

PRRT Predictably Reliable Real-time Transport. iii, v, vi, 3–7, 19, 23, 25–37, 39–41, 44, 45, 47, 48, 50, 54, 65–67, 69, 70, 74, 76, 81, 101–104, 117, 119, 124, 127–130, 133–135

PTP Precision Time Protocol. 44

QoE Quality-of-Experience. 22, 26

QoS Quality-of-Service. 2, 98

QUIC Quick UDP Internet Connections. 3, 21, 22, 25, 27, 43, 69

RACK Recent Acknowledgment. 68

RAM Random Access Memory. 130

ReLU Rectified Linear Unit. 72, 79

RI Redundancy Information. 3, 4, 17, 22, 36–38, 41–44, 50, 54, 56, 57, 63, 68, 69, 75, 77–80, 83, 84, 86–88, 93–95, 97, 98, 103, 112, 113, 117, 118, 124, 125, 128, 133–135, 139

RL Reinforcement Learning. 96

RLNC Random Linear Network Coding. 69, 128–131

RTP Real-time Transport Protocol. v, 22, 25, 27

RTT Round-Trip Time. 4, 16, 17, 20, 22, 31–34, 38, 43, 49, 50, 65, 68, 69

SACK Selective Acknowledgment. 68

SHARQ Scheduled Hybrid Automatic Repeat reQuest. iii, v, vi, 5–7, 52, 53, 55, 59, 60, 62–67, 69–71, 74–78, 80, 88, 90, 92–95, 99, 104, 108, 124, 134

SIMD Single Instruction/Multiple Data. 91, 127

SRT Secure Reliable Transport. 22

SRTP Secure Real-Time Protocol. 27

TCP Transmission Control Protocol. v, 3, 12, 15–17, 19–22, 25, 27, 31, 42, 43, 68, 97, 130, 135

TIMELY Transport Informed by MEasurement of LatencY. 20

TinyML Tiny Machine Learning. 74, 96, 129

TLP Tail Loss Probe. 68

TLS Transport Layer Security. 27

TPU Tensor Processing Unit. 74

TSN Time-Sensitive Networking. 12, 25, 135

UART Universal Asynchronous Receiver-Transmitter. 9

UDP User Datagram Protocol. v, 3, 12, 22, 25, 27, 119

URLLC Ultra-Reliable and Low-Latency Communication. 25

USB Universal Serial Bus. 119

VoD Video-on-Demand. 2, 26

VoIP Voice over IP. 2

X-Pace Cross-Layer Pacing. 27, 29, 33, 39

XOR Exclusive or. 69, 102–106, 108, 112, 113, 118, 127

Symbols

C Channel capacity. 21

D_T Delay target. 37, 40, 41, 45–51, 53, 54, 61, 62, 66, 69, 75, 77, 83, 126, 129

D_{PL} Packet loss detection delay. 30–32, 39, 41, 43, 54, 83

D_{RS} Sender and receiver processing delay per packet. 32, 39, 41, 51, 53, 54, 83

D_{tx} Transmission delay. 39

I_k Random variable for the number of lost data packets. 39, 40, 108, 112, 137, 138

L_p Packet length. 33, 40, 41, 54, 62, 83, 103–106

M Number of receivers in a multicast group. 38, 56, 58, 109, 110, 112

N_C Number of repair cycles. 32, 37–40, 45–47, 50–60, 62, 69, 75–77, 79–81, 89, 90, 96, 118, 124, 139

N_P Repair schedule. 32, 37–41, 50–52, 54–60, 69, 76, 77, 79, 96, 118, 119, 124, 139

$N_{C,max}$ Maximum number of repair cycles allowed by the delay constraint. 50–52, 58, 59

$N_{C,opt}$ Optimal number of repair cycles that minimizes the transmitted Redundancy Information. 54–56, 60

PLR_T Target packet loss rate. 37, 40, 41, 45, 46, 53, 54, 56, 60–62, 69, 77, 83, 112, 113, 115, 117, 126

RTT Round-Trip Time. 31–33, 39–41, 43, 45, 46, 50, 51, 53, 54, 62, 65, 66, 77, 83, 124

R_C Channel data rate. 33, 37, 40–42, 45, 46, 50, 54, 62, 77, 83

T_c Channel coherence time. 47, 48

T_r Protocol reaction time. 47–50, 65, 66

T_s Source packet interval. 30, 31, 38–41, 45, 46, 51, 53, 54, 62, 77, 83

Δ_{rr} Repair cycle interval. 31, 32

Δ DeepSHARQ's RI increase factor. 77, 82, 83, 86–89, 91, 93, 95, 96

α Field element. 32

β_0 Linear least squares population parameter. 122, 124

β_1 Linear least squares population parameter. 122, 124

$\boldsymbol{\theta}$ Neural network vector of learnable parameters. 71, 73, 74

$\boldsymbol{w}$ Vector of probabilities of cycles failing. 78

$\boldsymbol{x}$ Neural network vector of inputs. 71, 74, 77, 80

δ Binary matrix decoding failure probability. 106–109, 114, 115

λ Loss detection timeout. 30, 31, 39

$\mathbf{G}$ Generator matrix. 18, 19, 34, 101, 102, 105, 106, 111, 112

$\mathbf{c}$ Codeword vector. 18, 34, 51, 52, 101

$\mathbf{m}$ Message vector. 18, 34, 101, 102, 105

$\mathbf{n}$ Cumulative codeword length vector in an HARQ scheme. 38, 55, 57, 78, 108, 139

$\mathcal{C}$ Code. 18, 80, 102, 115

θ Neural network parameter. 72

ε Number of excess packets. 107, 109, 113, 114

b CPU bit width. 105, 106

c Free parameter of the Robust Soliton distribution. 109

d_{min} Minimum distance between codewords. 102

d Binary matrix column degree. 105–109, 114

e Number of erased symbols. 33, 40, 102, 103, 107, 114, 115

f_D Maximum Doppler frequency. 47

k Block length. 18, 19, 32–34, 37–41, 45–47, 50–63, 69, 75–83, 88–90, 96, 101–109, 111–115, 118, 119, 122, 124, 125, 127, 130, 137–140

$lossCount$ Event count in loss detection algorithm. 30

$loss\_threshold$ Loss count threshold for loss detection. 30, 31, 39

m Number of bits per symbol in the Galois Field. 18, 33, 57, 80, 81, 103, 104

n Codeword length. 18, 32, 33, 40, 57, 101–103, 106–111, 114, 115, 117, 118, 138, 139

p_e Channel erasure rate. 21, 33, 34, 38, 40, 41, 45, 46, 50, 54, 57, 60, 62, 63, 77, 79, 83, 108, 110, 111, 113–117, 137–140

p_x Probability distribution of the random variable X. 21, 81, 82

p Number of parity packets. 18, 19, 32–34, 37–40, 45–47, 50–62, 69, 75–81, 89, 90, 96, 104–108, 112, 114, 115, 118, 122, 124, 137–140

q_x Probability distribution of the random variable X. 81, 82

q Galois Field order. 18, 32, 33

r Code rate. 21, 106

Chapter 1

Introduction

These days, we live in a world where digital technologies seem to permeate every aspect of society. The rapid increase in computing, storage, and communication capabilities drives this wide adoption of digital technologies. Low-cost hardware proliferates thanks to improved manufacturing processes and the introduction of economies of scale, enabling developers to implement a myriad of digital applications. The dynamic software sector that has flourished in the last decades has arguably been a product of the available infrastructure in terms of open communication protocols and operating systems. However, this digital world is still limited to domains with an intensive information processing component—e.g., management in the firm and the government, retail, or entertainment. A similar level of adoption has not yet occurred in critical infrastructure such as transportation, logistics, heavy industry, and health care. The slow-paced adoption in these sectors could be explained by the higher dependability demands that led to using expensive, special-purpose hardware and software tailored for the task instead of cheap commodity components.

Cyber-Physical Systems (CPSs) have been proposed to bridge the gap between the digital (i.e., cyber) and the physical world by closing control loops monitoring physical processes over an infrastructure of embedded devices and networks. Edward A. Lee [54] claims that the applications resulting from coupling the digital and the physical world would have a far greater impact on society than the digital revolution did. However, these two worlds are qualitatively different. Time is inexorable in the physical world, and concurrency is intrinsic to it, while precise timing in current software is a challenging problem, especially in the presence of concurrency, which introduces intractable behavior. Moreover, CPSs impose safety and reliability demands on digital components, which they are not designed to uphold. The abstraction layers in the computing and networking substrate must be redesigned to achieve the required dependability level.

Similar to how the Internet has become the de facto standard for any communication, ruling out previous infrastructure—e.g., telephony, television, or radio—, Cyber-Physical Networking (CPN) could enable distributed CPSs over IP networks by fulfilling their specific performance needs. Such a deployment would only be possible due to the versatility of the Internet protocols, which create a layer of abstraction presented to a

myriad of applications over a common infrastructure, thereby diminishing the cost of deploying new services on top of the existing infrastructure. According to David D. Clark [137], this decoupling between network infrastructure and protocols allows us to envision many *future Internets* and, in particular, an Internet in which the requirements of CPSs for predictable reliability and delay are guaranteed.

1.1 Problem Statement

The demand for lower delays has been ever-increasing in the last decades, with the Internet infrastructure going from purely serving time-insensitive web content to more demanding services such as Voice over IP (VoIP) [77, 121], Video-on-Demand (VoD) [81, 126, 155, 183], or real-time video conferencing [110, 121, 197]. The appearance of CPSs has pushed the need for timeliness to unprecedented limits [54], as they close control loops over communications networks that connect sensors, actuators, and controllers. For CPSs, it is no longer enough to run tasks as fast as possible, but running them at the right point in time is key to ensure the control loop stability [48, 136, 177], which represents a paradigm shift from traditional trends in computer and network design.[1] Achieving the desired Quality-of-Service (QoS) on top of the unpredictable substrate the current operating systems and protocol stacks provide has proved difficult. From a purely information theoretical standpoint, delay-constrained data transmissions are bound to be unreliable [13, 70]. CPSs can operate on top of unreliable communication channels, as they are resilient towards deadline misses [30, 124, 177, 196], whether they are in the form of packet losses or late packets. Therefore, i) reliability and ii) timeliness become *functional requirements* the communication protocol stack must ensure to support CPSs.

CPS deployment is framed in what is known as the Internet of Things (IoT) [105]: inter-connected devices exchanging information in real-time to improve their operation in several sectors—e.g., agriculture, transportation, or industry. In other words, CPSs are expected to permeate our societies within the next decades. Given the sheer number of devices to be deployed, sustainability aspects should be considered in such a deployment. According to the 2023 report from the Intergovernmental Panel on Climate Change (IPCC) [219], a "rapid" and "deep" reduction of greenhouse gas emissions across all sectors and systems is required before 2030 to limit global warming to 1.5°C or 2°C, thereby reducing the heat-related hazards for life on the planet. CPSs play a key role in fulfilling such requirements—e.g., in the deployment of smart grids to reduce the CO_2 footprint of energy supply [71] —, but a sustainable deployment is only possible by improving their energy efficiency [34, 55, 199, 216]. Energy efficiency is a *non-functional requirement* that conflicts with the two aforementioned functional requirements—i.e., redundancy information is required to recover from packet losses, increasing the power draw. The system's energy demand should be reduced if possible so that the system's resource footprint is minimized [5, 111].

[1] See the trend towards higher throughput and lower delay with each new generation of mobile communication, or increasing CPU frequency and number of cores.

1.2 Research Question

In IP networks, the transport protocol provides the communication entry point to the application via *sockets*, and hence, it is the lowest layer with an end-to-end perspective of the communication channel as perceived by the application. Following the *end-to-end argument* in systems design [18], the transport layer is key to fulfilling the application requirements. Therefore, this layer must implement the mechanisms to provide performance guarantees to the application—or inform the application whenever meeting the requirements is impossible. For example, since web content request and delivery is based on Hypertext Transfer Protocol (HTTP), the World Wide Web revolution would arguably not have been possible without the Transmission Control Protocol (TCP) that HTTP uses as transport protocol.[2] The superiority of TCP over other protocols has resulted in its adoption even for real-time applications that it was not originally designed to support [81] as a result of its wide availability, since most Operating Systems (OSs) implement it nowadays to provide web browsing and most firewalls admit it for the same purpose, which is not the case for other transport protocols.[3] If the wide adoption of web applications has been fostered by the broad availability of TCP, it stands to reason that the widespread use of CPSs will also require a transport protocol that provides a simple-to-use interface on top of which developers can implement CPSs. The objective of this thesis is to answer the research question:

Research Question: *how should transport protocols be designed to provide broad support for Cyber-Physical Systems?*

According to the problem statement above, what CPSs demand is predictably reliable, real-time communications that at the same time are energy-efficient. The Predictably Reliable Real-time Transport (PRRT) protocol [57, 86, 163] seems a suitable candidate: it is a time-aware protocol that takes the target delay and reliability level from the application and adapts its performance to fulfill them. The fundamental transport layer function providing the predictable performance is the *error control* function: it transmits the minimum amount of Redundancy Information (RI) required to recover lost packets, thereby achieving the desired reliability level within the application time budget.

The reactive Automatic Repeat reQuest (ARQ) is the most used error control scheme as it is implemented in TCP. It implements a feedback mechanism to trigger packet retransmissions when losses are detected. So long as packet losses can be perfectly detected, the sender transmits the minimum required RI to recover the losses. Nevertheless, a proactive scheme may be a more suitable option if: i) there is not enough time to wait an entire round-trip time for feedback, ii) there are so many receivers in a multicast group that the feedback causes what is called *feedback implosion*, or iii) a feedback chan-

[2] It has not been until late 2022 that a version of HTTP was standardized [200] to support a different transport protocol, namely the Quick UDP Internet Connections (QUIC) protocol.

[3] As more web browsers implement the QUIC protocol, it is expected that firewalls will stop filtering the User Datagram Protocol (UDP) as QUIC connections are deployed on top of it.

nel is not available. The proactive Forward Error Coding (FEC) encodes *parity packets*, linear combinations of data packets that can be used at the receiver to recover packet losses. Parity packets can be encoded without waiting for feedback from the receiver about lost packets to transmit redundancy proactively. Under delay constraints, the delay of the reactive ARQ depends on the Round-Trip Time (RTT), while FEC's delay depends on the source packet interval of the application that is required to collect data packets before encoding. The different delay nature of these two approaches makes the combination of the two, which Hybrid Automatic Repeat reQuest (HARQ)[4] implements, ideal for delay-aware error control. This thesis looks into two different components of delay-aware error control to answer the aforementioned research question:

- **C1 Optimizer**: finds the optimal HARQ configuration that minimizes the transmitted RI while meeting the application delay and reliability constraints. In a dynamic environment, it should react to network changes fast enough to avoid blackout periods in which the fulfillment of the constraints is not guaranteed.
- **C2 Coder**: en-/decodes parity packets to recover from losses. The coding step—especially the decoding part, as it performs a matrix inversion and matrix-vector multiplication while encoding only performs the latter—is typically a computationally expensive process, whose realization depends on the implemented coding technique [26, 72].

To answer the aforementioned research question, this thesis looks at these two components implemented in PRRT through the CPS glasses, thereby making original contributions to the field of CPN.

1.3 Contributions

This section introduces the theoretical analysis, empirical evaluations, and practical implementations resulting from the research question above, which are all crucial contributions to the *Energy-, Latency- And Resilience-aware Networking (e.LARN)* project.[5] The implemented code is available as Rust libraries to facilitate its deployment in other projects, and has already been integrated into PRRT's source code.[6]

Predictably Fast Reaction to Channel Changes

For PRRT to provide the application with a predictable communication channel, it must react in a timely manner to changes in the underlying dynamic channel to ensure the

[4]The terms HEC and Hybrid Automatic Repeat reQuest (HARQ) have been used in the literature to refer to the combination of FEC and ARQ into a single error correction scheme. In 5G [173], the term HARQ refers to a coded version of ARQ that does not use the proactive FEC cycle. In opposition to this definition, HEC could be understood as a scheme that also uses the proactive cycle. Nevertheless, we do not make such a distinction here and use both terms interchangeably.

[5]`https://www.nt.uni-saarland.de/projects/elarn/` (accessed January 23rd 2024)

[6]`https://git.nt.uni-saarland.de/LARN/PRRT` (accessed January 23rd 2024)

constraints are always fulfilled. This thesis provides a computational complexity analysis of the optimization algorithm that finds the optimal HARQ configuration meeting the constraints. Due to the lack of a closed-form expression to obtain the optimal configuration, computationally expensive search algorithms have been employed so far [86]. Two different approaches are presented in this thesis that improve the predictability of the communication channel: i) SHARQ is a purely algorithmic approach that, thanks to its more efficient solution space exploration, reduces the computational complexity of finding the optimal configuration, and ii) DeepSHARQ combines the algorithmic optimizations in SHARQ with deep learning algorithms and a novel solution space regularization mechanism to construct small neural networks, thereby bringing predictable communication channels to resource-constrained embedded devices, the natural component of CPSs, for the first time.

Energy-Aware Redundancy Coding

PRRT's redundancy is generated with Vandermonde codes that solve a linear equation system in high-order Galois Field to en-/decode parity packets. This process was originally assumed to occur infinitely fast, as the delay and energy demand are negligible on high-end computers. However, this assumption does not hold when the protocol is deployed on resource-constrained devices due to the high complexity of the matrix-vector multiplication performed to solve the equation system. This finding led to the reformulation of the *complexity dilemma* that has guided the development of codes for the last decades: in the transport layer, the matrix inversion does not govern the coding complexity, but the matrix-vector multiplication does. To the best of our knowledge, this thesis presents the first systematic analysis of the suitability of binary codes as an alternative to Vandermonde codes in the transport layer. Binary codes implement a more efficient matrix-vector multiplication, at the cost of a redundancy overhead. In other words, the processing complexity is reduced at the expense of the network complexity. Based on energy and delay models constructed for the coding function running on a Raspberry Pi Zero W, we show that binary codes are a more energy-efficient alternative in the transport layer for a wide range of applications, especially as the delay constraint becomes more stringent.

1.3.1 Summary

A summary of the contributions of this thesis, and how they interact within PRRT's error control function, is depicted in Fig. 1.1. The contributions, highlighted in red in the figure, are qualitatively compared in terms of their ability to adapt to CPS demands for predictable reliability and timeliness, as well as how they use the available computing and networking resources. Two different optimizers have been proposed, namely SHARQ and DeepSHARQ, whose main difference is the achieved reliability level and their computational complexity. For an arbitrarily small subset of channel conditions, SHARQ can find suitable configurations but DeepSHARQ does not. This suboptimality is a result of the employed deep learning method, which at the same time has the benefit

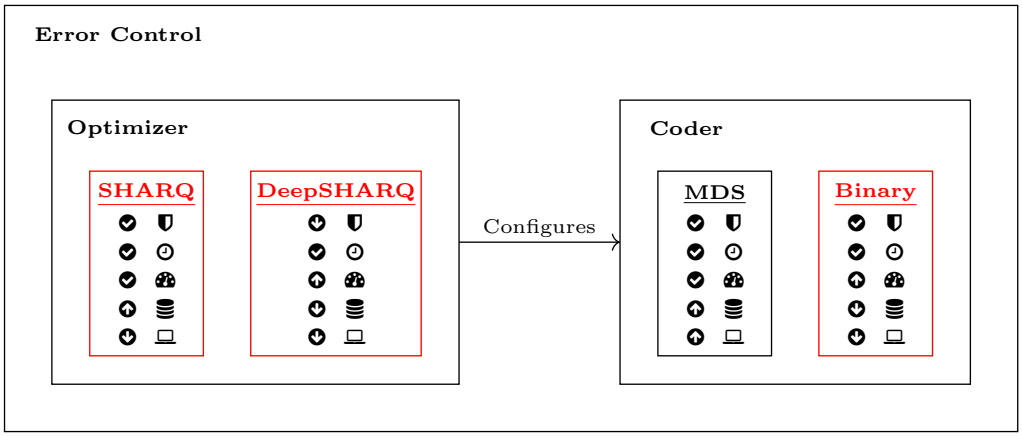


Figure 1.1: Summary of the Contributions of this thesis (highlighted in red). The boxes describe how every contribution impacts the reliability (🛡) and timeliness (🕐) requirements of CPSs, as well as the throughput (🚀), memory (🗄) and computational complexity(💻). The circles next to each element compare them to alternative components in terms of higher (🔼), lower (🔽), or optimal (✅) values.

of reducing DeepSHARQ's computational complexity. As a result, the optimizer can be brought to low-end devices on which SHARQ takes too long to infer the configurations. Another aspect to consider is the memory footprint of the algorithms because SHARQ has a large memory footprint, while DeepSHARQ's memory footprint is two orders of magnitude smaller. As will be discussed later in the thesis, CPSs may have limited resources at their disposal, whether it is throughput, memory or processing power, or any combination of the three. The trade-offs between the two proposed algorithms allow PRRT to adapt its performance between optimum throughput on high-end devices (SHARQ) or a throughput overhead but reduced memory and computational complexity (DeepSHARQ). A similar trade-off is present in the coder that is configured by the optimizer. PRRT originally implemented the optimal MDS codes, which produce the optimal amount of redundancy at the cost of high computational overhead. The binary coding library analyzed in this thesis shows that again, by giving up optimality in networking resources, the memory and computational resources employed by the protocol can be reduced, thereby reducing the energy demand of the system at the same time.

1.4 Outline

This thesis can be split into two differentiated parts: Chapters 2 and 3 lay the foundations of this work, while Chapters 4, 5 and 6, present the original contributions of this thesis, which focus on the optimization algorithm finding HARQ configurations (Chapters 4 and 5) and energy-awareness for PRRT's coding function (Chapter 6).

First, Chapter 2 introduces CPSs, their domain applications, and the requirements that impose strict constraints on the underlying communication and computational sub-

strates. The chapter discusses the transport layer's central role in fulfilling those requirements so that CPSs could be widely adopted. A literature review of timeliness in the transport protocol, followed by a discussion of the theoretical limits of delay-constrained communications closes this chapter. Chapter 3 begins with a literature review of timeliness in the transport protocol, followed by a discussion of the theoretical limits of delay-constrained communications. The chapter then presents the PRRT protocol, including a detailed description of its two main transport layer functions, namely cross-layer pacing and adaptive HARQ. The chapter ends with the formal definition of the optimization problem PRRT solves to find the optimal HARQ configuration that meets the application constraints.

Second, Chapter 4 begins with the theoretical framework used in the thesis to compare search algorithms by assessing their ability to provide predictable communication channels to the application. The chapter also presents the SHARQ algorithm, which has been optimized to achieve a high predictability level on high-end computers but has failed to achieve similar predictability on resource-constrained devices. This poor performance motivated the DeepSHARQ algorithm presented in Chapter 5, which is based on a novel output space regularization technique that reduces the complexity of the learning problem. Combining this learning technique with the algorithmic optimizations presented in the previous chapter and a simple parity packet scheduler results in significantly shorter inference times, thereby bringing predictable communication channels to resource-constrained devices for the first time.

Third, Chapter 6 revisits the complexity dilemma when designing coding techniques that must balance processing and network complexity for redundancy generation and transmission. Several alternative codes are presented that could improve the energy efficiency of the coding function in PRRT. The chapter then continues with a theoretical and practical analysis of these alternatives, to finish with empirical proof that binary codes are an energy-efficient alternative for a significant number of application scenarios, but not enough cases to rule Vandermonde codes out.

Finally, Chapter 7 concludes the thesis with a summary of the contributions and provides directions for future research that would complement the results presented in this thesis.

Chapter 2

Distributed Cyber-Physical Systems

CPSs have the potential to revolutionize the world as we know it by bridging the gap between the digital and physical world. Nevertheless, for CPSs to reach their full potential, they should flourish in the right ecosystem, offering the software and hardware infrastructure for developers. This chapter formally defines CPSs, their requirements, and application domains. Moreover, this chapter makes the case for the transport layer as the key component in the protocol stack to enable the kind of communication that CPSs demand for their correct operation.

2.1 Cyber-Physical Systems

According to Lee [116], the term *cyber-physical systems* was first introduced in 2006 by Helen Gill at the National Science Foundation in the United States of America. The root of this term can be found in the closely related field of *cybernetics*, which in the second half of the 20th century grouped the fields of control and communication theory. The term cybernetics was coined by Norbert Wiener and it comes from the Greek $\chi\nu\beta\epsilon\rho\nu\acute{\eta}\tau\eta\varsigma$ or *steersman* [14], an appropriate metaphor for the control systems he studied, whether in the machine or the animal. Although Wiener did not use digital computers, the control logic is effectively a computation, hence the relation to CPSs, as they are both a conjunction of digital and physical processes.

Definition 2.1.1 (Cyber-Physical System)**.** A Cyber-Physical System (CPS) integrates computation and physical processes using embedded computers and networks that monitor the physical process with sensors and control it with actuators [54].

CPSs differ from other systems in their demands for predictable timeliness and reliability, as discussed in Sec. 2.1.2. Traditional CPSs embed all their components into a single chassis, thereby simplifying their design due to the dedicated access to resources. The communication in these systems is performed via Inter-Integrated Circuit (I2C) or Universal Asynchronous Receiver-Transmitter (UART) for synchronous and asynchronous,

respectively. These communication technologies provide a predictable communication substrate due to the short distances between components and the dedicated access to the medium. However, some CPSs are distributed in nature (see Sec. 2.1.1 below), such that sensing, actuation, and computation may be performed at completely different locations. For these systems to leverage the existing Internet infrastructure, a paradigmatic change is required from the throughput-oriented communication that is currently seen in data-intensive industries—e.g., web-browsing or social media—, to a real-time-oriented design with performance guarantees. The term Cyber-Physical Networking (CPN) was coined to group the technologies developed under this umbrella.

Definition 2.1.2 (Cyber-Physical Networking)**.** Cyber-Physical Networking (CPN) refers to the set of communication protocol standards and implementations, whether in hardware or software, that enable communication in distributed CPSs with their real-time guarantees.

The formal definitions above may seem too general, but they contain all the fundamental building blocks required for digital systems to interact with the physical world, while changes in the physical world alter the state of the digital model simultaneously. Based on these definitions, the list of application domains whose capabilities could be enhanced by applying CPSs seems endless.

2.1.1 Application Domains

The list of application domains includes, but is not limited to, logistics, healthcare and medicine, transportation, manufacturing, energy generation, agriculture, or critical infrastructure control. As information-processing components are embedded within these domains, their capabilities could be enhanced to achieve a more rational control of the social metabolism.[1] Given the time of climate change in which we currently live, as described in Sec. 2.2, the term *rational* is inevitably linked to the concept of *sustainability*. The following examples illustrate the potential of CPSs to improve efficiency in a broad set of processes, thereby enabling sustainable development on a global scale. Traffic control could be improved with autonomous vehicles exchanging information among themselves and with roadside infrastructure for a more efficient and safer transportation system. This infrastructure could also be leveraged by the logistics sector, which is currently experiencing a revolution driven by swarm robotics. A better healthcare system could be possible thanks to precise medical devices and telemedicine, which could bridge the gap between urban and rural areas, especially in remote areas. The real-time control of critical infrastructure—i.e., power, water, and communication grids—could result in better waste management and the integration of distributed green energy generation. Precision agriculture could monitor crops to optimize the nutrients and pesticides employed. Manufacturing automation could be brought to the next level with the cen-

[1] By social metabolism, here we mean all the flows of materials and energy between nature and society so that a society can reproduce itself. Although the term dates back to the 19th century, it was not until the late 20th century and early 21st century that it gained relevance with the surge in interest in environmental science [98].

tralized digitization of the end-to-end chain, including real-time information on product demand and supply chains.

Judging by these examples, CPSs are related to other currently popular trends in the industry and the scientific community [85, 101, 105], such as IoT, Industry 4.0, fog computing, digital twins, or the Tactile Internet. However, we believe CPS is a more general term that does not refer to particular approaches or implementations but to the fundamental challenges that arise in the intersection of the cyber and physical worlds. These challenges originate from their requirements in terms of predictable reliability and timeliness in order to ensure their safe operation.

2.1.2 Requirements

The examples above show that CPSs are expected to interact with safety-critical infrastructure. Failing to operate in such environments correctly may result in unrecoverable losses of equipment and lives or put entire countries at risk. Moreover, the physical world is inherently unpredictable, and thus CPSs must be robust to unexpected conditions that could not have been foreseen at the time of their design. In other words, *timeliness* and *reliability* are functional requirements of CPSs—i.e., they define the correct behavior of the system rather than being a simple performance metric.

Reliability

Advances in computing and communication techniques have resulted in highly reliable systems able to store trillions of bytes (memory), perform billions of operations per second (processors), or transmit billions of bits over dynamic channels in the time and frequency domains (network card). Nevertheless, since no component is perfectly reliable, the probability of at least one failing becomes high as the number of components drastically increases to build complex CPSs. In such heterogeneous systems, failures can have different origins, e.g., sensors may run out of battery, packets with critical information may be dropped in the channel, and concurrency may introduce non-deterministic behavior in the software. Luckily, the control loops CPSs rely upon are not *hard* real-time systems that would fail as soon as a deadline fails or data is lost. These control loops can be modeled as *weakly hard* real-time systems instead [30]: the stability of the control loop can be guaranteed in the sense of Lyapunov as long as deadline misses [124] and data losses [122] occur predictably. These theoretical results have been confirmed in practice as well [196, 209, 224]

Robustness could also be extended to other abstraction layers in system design. Lee [54] proposes the following principle: “when feasible, components at any level of abstraction should be made predictable and reliable. Otherwise, the next level of abstraction above these components must compensate with robustness”. This principle is just another version of the *end-to-end* principle [18] that is discussed in more detail in Sec. 2.3.

Timeliness

In the physical world, time is inexorable, and concurrency is an intrinsic feature. However, although these properties are present in current computing and networking systems, they are lost in the abstraction layers currently used in systems design [54]. Despite being deployed on a nearly perfectly predictable substrate, namely digital circuits, current software does not have access to high-level abstractions that leverage such predictability [54]. CPU instruction sets, programming languages, or operating systems do not include timing semantics and when they do, they are inherently imprecise—see timers in operating systems [180]. A similar scenario occurs with the networking components: wireless protocols have precise symbol durations and implement a distributed synchronization mechanism to detect transmissions correctly. This information is kept within the network card, and operating systems do not forward it to the upper layers of the protocol stack. In the wired domain, Time-Sensitive Networking (TSN) [139] and Deterministic Networking (DetNet) [149] provide deterministic layer 2 and 3 protocols. TSN provides synchronization mechanisms for all the devices in the network to have a shared sense of time, thereby executing the required operations at the precise point in time. Unlike Internet protocols, which are *best-effort*, TSN reserves throughput and transmission slots for low latency applications not to spend time buffering. DetNet extends the same rationale one layer above in the protocol stack. Nevertheless, Transmission Control Protocol (TCP) [205] and User Datagram Protocol (UDP) [17], the two de facto transport protocols for reliable and unreliable data transmission, respectively, do not provide any deterministic guarantees (see Secs. 2.3.2 and 2.3.4).

2.2 Energy, Power, and Sustainability

The concern over climate change has significantly grown in the last decade, positioning it as one of the major challenges for the 21st century. To prevent climate change's most pernicious effects, policies are already being put in place at a global scale, which will likely impact a pervasive technology such as CPS. As an introduction to the causes and effects of climate change, this section considers the reports published by the Intergovernmental Panel on Climate Change (IPCC). As a body dependent on the United Nations, the IPCC assesses the science related to climate change to provide governments with useful information for developing climate policies. Its members are volunteers reflecting a range of scientific, technical, and socio-economic backgrounds selected to avoid bias toward the views of a single country or groups of countries. These members review scientific papers published each year on topics related to climate change to analyze the scientific consensus on the area. Such an analysis results in the Assessment Reports published over the years. This section has been based on the sixth report published in 2023 [219].

The report is crystal clear in pointing out that "human activities, principally through emissions of greenhouse gases, have unequivocally caused global warming". Particularly, the focus is put on three greenhouse gases as the major contributors to global warming, namely carbon dioxide (CO_2), Methane (CH_4), and nitrous oxide (N_2O).

The accumulation of these gases in the atmosphere has produced an increase in global surface temperature of approximately 1.09°C by 2011–2020 compared to the temperature in 1850–1900. This increase has already affected the weather and climate in every region globally, reducing food and water security and increasing extreme heat events, resulting in mortality and morbidity. There is evidence of changes in extreme weather, such as heatwaves, heavy precipitation, droughts, and tropical cyclones. Researchers are highly confident that climate change has already increased food-borne, water-borne, and vector-borne diseases. Further global warming would make these risks "increasingly complex and more difficult to manage".

Estimations show that in 2019 79% of greenhouse gas emissions came from energy, industry, transport, and building sectors, whereas 22% of these emissions came from agriculture, forestry, and other land use.[2]

2.2.1 Sustainable Cyber-Physical Systems

There is a clear overlap between the aforementioned sectors contributing the most to greenhouse gas emissions and the application domains for CPSs in Sec. 2.1.1. This overlap shows the potential of CPSs to enable more efficient processes in our society by reducing their carbon footprint.[3] However, CPSs come hand in hand with a large-scale deployment of sensors, actuators, and new computing power, which demand energy and resources. Failing to address the causes of climate change in time could turn the Earth into an inhospitable planet for human beings. Therefore, ensuring a sustainable deployment of CPSs is key from an environmental standpoint. Hence, energy efficiency becomes a requirement that must be considered when designing and deploying these systems.

Energy Efficiency

Energy efficiency is a conflicting requirement with the aforementioned reliability and timeliness requirements: the generation and transmission of parity packets to reach the desired reliability level presupposes a power draw excess from the system, as do the channel and system monitoring code to implement awareness. While reliability and timeliness are *functional* requirements governing the correctness of the system's operation, in safety-critical applications, energy efficiency is a *non-functional* requirement as it is a desired property, but it should not be enforced to the detriment of correctness.

[2] The authors of the report point at rounding errors aggregating the data for these digits not adding up to 100%.

[3] The IPCC report highlights that, despite the improvements in energy intensity of gross domestic product and carbon intensity of energy, they do not offset the increase in global activity levels. This behavior resembles the *rebound effect* theorized by Jevons in [12], whose hypothesis was about the diminished gains of technological improvements leading to more efficient use of resources due to behavioral or societal responses. Nevertheless, Ostrom [21] showed that humans can build societal institutions that efficiently administer shared resources, not always mediated by markets but by sophisticated decision mechanisms and compliance with standards. All in all, even though CPSs could help mitigate the effects of climate change, the solution to this problem will probably not be purely technical but societal.

In other words, energy-awareness should be implemented to reduce the energy demand whenever possible while ensuring its correct operation.

For a sustainable deployment, the energy efficiency of the system should be holistically studied, including the low-end devices for sensors and actuators at the edge, the data center running the control algorithms either in the edge (i.e., fog) or deeper in the network (i.e., cloud), and the communication infrastructure bridging both ends. Most of the attention has been directed towards the energy efficiency of data centers [111, 140, 195, 186, 199], as they can consume a significant percentage of the electricity of a country [65, 222]. The interconnection of low-end devices has also drawn a lot of attention recently, especially with the design of protocol stacks that target their requirements [35, 37, 176, 204]—i.e., lightweight stacks with a low resource footprint and physical layer protocols designed to demand little energy. After some early research by Gupta et al. [34], the Internet infrastructure remains one of the least studied components, which [216] argues could be due to the "lack of awareness of the problem; a lack of standards for who should be responsible for collecting and attesting to what data; and a lack of tools for collecting data." [216] highlights the need to go beyond energy efficiency by considering the carbon footprint as well.

Scarcity of Resources

As described above, sustainability concerns make it imperative to reduce the CO_2 footprint of CPSs, whether these emissions come from the system's energy demand—i.e., power grid—or the materials it is made of—i.e., mining, manufacturing, and transportation systems. In the last decades, the deployment of the most performant applications has been driven by the exponential growth in the capabilities of digital systems in terms of processing speed, memory size, and throughput, together with a reduction in production costs. As this option is not available to CPSs, the most challenging aspect in their deployment is that their stringent performance demands must be achieved while reducing the resource footprint of the systems. System designers should take into account the reduced availability of resources such as the transmission and reception data rates, memory size, CPU clock speed, or battery capacity.

For example, the physical components, such as sensors and actuators, may have constraints in terms of accuracy, resolution, or responsiveness to reduce their deployment costs. Hence, control algorithms must be robust enough to operate with these low-accuracy samples, frequently obtained at a lower sampling rate than if resources were abundant. Communication for IoT devices trade energy efficiency for lower throughputs than in standard WiFi and 5G. Secure communication protocols, cryptographic operations, and authentication processes must be redesigned to work on such resource-constrained devices. Consequently, the advancement and cooperation in fields such as hardware design, software and control engineering, telecommunications, or algorithms are crucial to CPSs.

2.3 The Centrality of the Transport Layer

As its name already suggests, the Internet is a set of interconnected computers exchanging digital information in the shape of *data packets*. At its core are the IP and TCP protocols. The former routes and forward packets so that the sender and receiver computers can communicate, and thus IP runs in the end nodes and intermediary nodes—i.e., routers. The latter is only executed in the end nodes and provides the mechanisms to achieve a fully reliable, in-order stream of bytes from the sender to the receiver, despite the bit errors and packet losses that may occur in the underlying network.

Maintaining the core specification—i.e., packet format, provided services, and main interfaces–of these two protocols unchanged has a double advantage: i) application designers only need to understand the interface with the protocol and forget about the technicalities below it, and ii) the designers of the underlying communication technologies—i.e., physical and link layers—must ensure they support these protocols but do not need to bother about specific applications. Clark [137] argues that this architectural design is behind the success of the Internet as a network, as it fosters a highly diverse ecosystem of applications on top of TCP (e.g., web, email, chatting, or video streaming) and a similarly diverse set of communication technologies below IP—e.g., Ethernet, broadband, WiFi, or cellular networks.

Another essential aspect of such widespread adoption of Internet technologies is the availability of open-source protocol implementations. The paradigmatic example is the web, with the HTTP protocol and open-source web servers available in the different distributions of the Linux OS.

2.3.1 Rationale

We ask ourselves whether a similar ecosystem could be built to encourage the wide deployment of CPSs, which would call for redesigning operating systems and network protocols as we understand them nowadays. This thesis focuses on networking aspects, but the reader interested in how CPS performance guarantees are implemented in operating systems is referred to [5, 180]. The current Internet architecture is a consequence of the *end-to-end* argument, a design principle formulated by Saltzer et al. [18] in 1984. According to this principle, the functions placed at the low levels of a distributed computer system may be redundant or incur a high cost. Although functionalities in the low levels could be justified as long as they provide performance enhancements, the functionalities should be brought as close as possible to the application because only the endpoints of the communication channel can ensure the correctness of the functionality. For example, ensuring fully reliable communication at the router level would require a costly information exchange mechanism to confirm the correct packet reception at every hop. Even if this mechanism were available, routers could not guarantee data reception once it is handed over to the transport layer, e.g., a flipped bit in the end node before it is forwarded to the application. Therefore, only the highest level—i.e., the transport layer in an IP stack—can provide such reliability guarantees to the application.

Following this principle, most Internet functionalities are implemented in TCP and hence the end-node only. The three fundamental transport layer functions implemented in TCP are *flow control*, *congestion control*, and *error control*. The relevance of these functions is reflected in the *cross-layer effects* [1, 89]: the algorithms implemented in the transport layer impact the application's performance, and thus, the underlying protocols should be kept in mind during the application design phase. Although deterministic networking is available in the lower layers [139, 149] and transport layer functions have also been deployed in routers to improve the protocol's performance [106, 130, 127], ultimately the performance perceived by the application depends on the transport protocol at the end node with which it has a direct interface.

Flow Control

If the sender's transmission rate is higher than the receiver's processing rate, some of the transmitted packets will be dropped at the receiving end, with the waste of resources it entails. The flow control function ensures that the sender does not overwhelm the receiver with more data than it can process [205]. TCP implements a sliding window mechanism in which the receiver announces how much data it is willing to accept. The sender uses this information so that the number of outstanding bytes—i.e., the transmitted bytes whose reception has not been acknowledged by the receiver yet—must be smaller or equal to the announced buffer size. The receiving buffer was a limiting factor in the early age of the Internet when memory was expensive and scarce. Although it has become a lesser issue due to the abundant cheap memory now available, slow-processing receivers may become the limiting factor in data transmission again in IoT scenarios with resource-constrained devices.

Congestion Control

Similarly to flow control, the sender may transmit data faster than the network can process, thereby producing network congestion. When a link is congested, packets are stored in a buffer in the router. Packet buffering increases the Round-Trip Time (RTT) of the link not only for the flow causing congestion but for all the flows sharing that same link as well. If congestion persists, the buffer eventually overflows, causing packet losses that need to be recovered since TCP is fully reliable. TCP implements a Congestion Control Algorithm (CCA) that detects congestion and adapts the sender's data rate to avoid congestion in normal conditions and recover from congestion when it is detected. CUBIC [53] is the default CCA in Linux, and uses buffer overflows as the congestion signal. However, recently proposed algorithms [64, 102, 108, 112, 134, 158] aim at avoiding buffer queues in order to reduce the RTT. These algorithms are further analyzed in Sec. 2.3.2.

Error Control

TCP implements an in-order, fully reliable bytes stream service. Therefore, packets must carry a sequence number indicating the position in the sequence of the first byte in it (a random sequence number is used in the first transmission in a connection), and the receiver transmits Acknowledgments (ACK) with the sequence number of the last in-order, received byte. The error control function specifies that the receiver must issue a duplicate ACK upon receiving an out-of-sequence packet. In the default mode specified in [59], the sender can detect losses upon a timeout or the reception of three duplicate ACKs. The latter allows for a fast recovery since it assumes the network still operates correctly, but at least one packet was dropped by a sporadic congestion event or some noise in the channel, whereas the former assumes a major breakdown due to factors such as persistent congestion or link failure, and hence begins the retransmission at a low transmission rate not to overwhelm the network until the failure is solved. More advanced error control schemes have been proposed in the literature to reduce the loss recovery time [63, 79, 93, 109, 187], which are described in more detail in Sec. 2.3.2.

Error Control Schemes

The ACK-based mechanism implemented in TCP is known as Automatic Repeat reQuest (ARQ). Such a reactive retransmission scheme makes the error control's delay proportional to the RTT elapsing between a packet transmission and the reception of its ACK.

Alternatively, a transport protocol could implement the Forward Error Coding (FEC) scheme, which does not require feedback from the receiver. FEC proactively transmits the Redundancy Information (RI) required to recover from losses. As no information about lost packets is available at the time of transmitting the redundancy, FEC must encode *parity packets*, which are a linear combination of data packets, so that losses can be recovered by solving a linear equation system at the receiver. As a result, the loss recovery delay is no longer RTT-dependent, but it is proportional to the source packet intervals that the sender must wait to collect packets before encoding.

From an information theoretical standpoint, an error control scheme is deemed optimal if, and only if, it minimizes the transmitted RI. ARQ and FEC fundamentally differ in two aspects: the nature of their delay (i.e., reactive vs. proactive) and the certainty about packet losses (loss detection vs. loss prediction). Therefore, it stands to reason that both approaches should be combined to provide optimal, predictable reliability under delay constraints so that the optimal balance between proactive (FEC) and reactive (ARQ) that minimizes the transmitted RI is found. Hybrid Automatic Repeat reQuest (HARQ) implements precisely that behavior: parity packets can be transmitted in the proactive or reactive cycles, and the sender stops transmitting redundancy when the receiver signals it has enough to recover the losses or until it is too late to recover them in time. Fig. 2.1 provides a graphical comparison of the three aforementioned schemes.

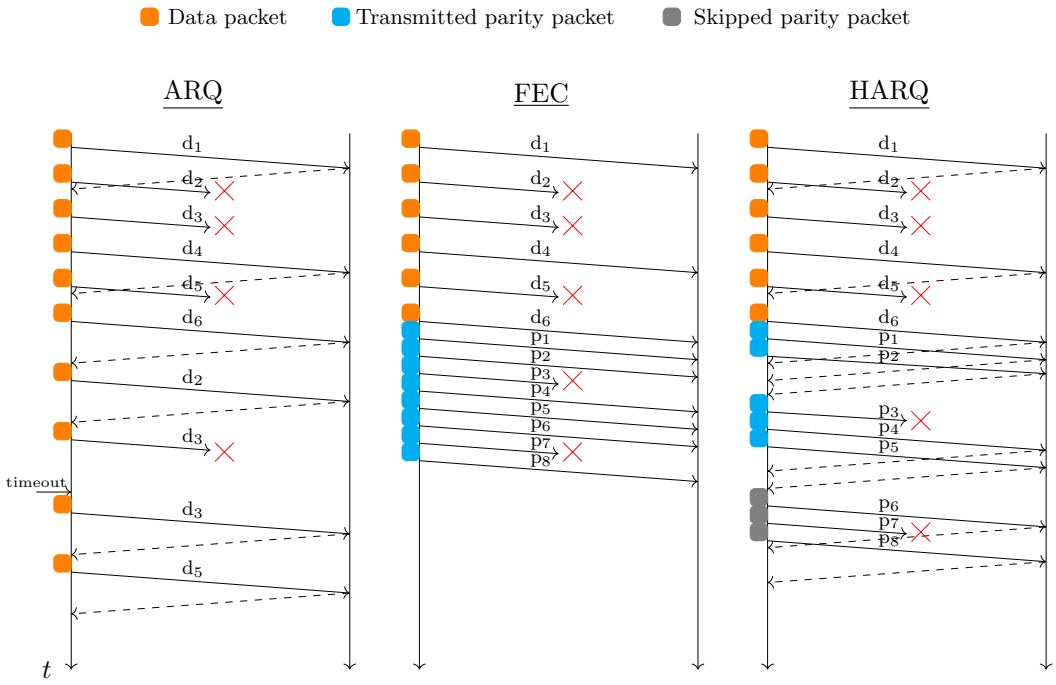


Figure 2.1: Comparison of the different redundancy transmission schemes for error control.

Parity Packet Coding

HARQ transmits parity packets—or, more generally, *parity symbols*—to recover losses. From all the available mechanisms (i.e., codes) to generate these parity packets, in this thesis we focus on *block codes*. For the curious reader interested in other coding techniques, a good introduction is available in [16].

A block code $\mathcal{C}(n,k) : \mathbb{F}_q^k \rightarrow \mathbb{F}_q^n$ transforms a message vector $\mathbf{m}$ into a codeword $\mathbf{c} \in \mathcal{C}$. The finite field $\mathbb{F}_q$ has a size q. Typically, the field is selected from the family of $GF(2^m)$ for binary representation, where m is the number of bits per symbol in the alphabet. Here, k is the block length—number of symbols in $\mathbf{m}$—, and n the codeword length—number of symbols in $\mathbf{c}$. The symbols are encoded by performing a matrix-vector multiplication with the generator matrix $\mathbf{G}$ ($\mathbf{c} = \mathbf{m} \cdot \mathbf{G}$). At the receiver, the original message vector is recovered by performing the inverse operation ($\mathbf{m} = \hat{\mathbf{c}} \cdot \hat{\mathbf{G}}^{-1}$). $\hat{\mathbf{G}}$ is a $k \times k$ submatrix of $\mathbf{G}$, whose columns have been selected based on the position of the received symbols $\hat{\mathbf{c}}$. Fig. 2.2 shows how the encoding operation is performed. We assume a systematic code is used—i.e., the $k \times k$ identity matrix is part of $\mathbf{G}$, and thus the codeword contains a verbatim copy of the message vector. Systematic codes reduce the coding complexity as only $p = n - k$ symbols are encoded instead of n, achieve better error correction capabilities: if the linear system cannot be solved—e.g., it is undetermined because fewer than k packets were received—, they can still forward

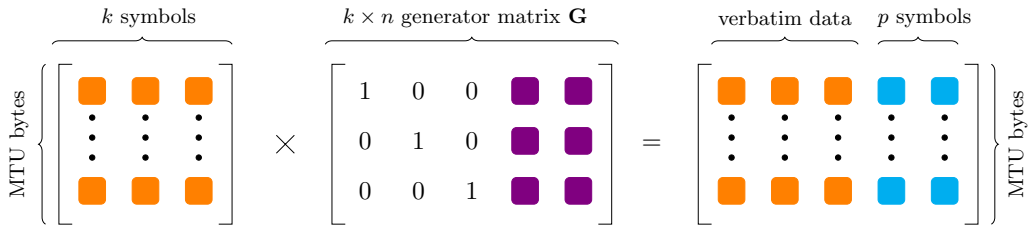


Figure 2.2: Encoding process of a systematic code with a block length k, p parity packets and a generator matrix $\mathbf{G}$. Symbols are packets of MTU bytes.

the received verbatim data without decoding, and they also allow for data transmission before all the k packets are collected for encoding, which reduces the end-to-end delay.

While the physical layer performs the coding operation at the symbol level—i.e., directly in bits—, IP networks are packetized erasure channels, meaning that full packets are lost in the network because packets with uncorrectable bit flips are not forwarded to the upper layer, or full packets are dropped due to buffer overflows. As a result, HARQ at the transport layer must be capable of recovering full packets. The length of IP packets is determined by the Maximum Transmission Unit (MTU) of the underlying physical channel, which in Ethernet is 1,500 bytes. A common solution to coding with large packets is *virtual interleaving*, which splits the packet into smaller symbols of m bits. The interleaver buffer collects k packets; hence, the coding operations are iterated throughout the complete packet length. PRRT implements precisely this coding mechanism in $GF(2^8)$ and is described in more details in Sec. 3.3.3.

2.3.2 Timing and Transport

Every new generation of wired and wireless communication technology has witnessed an increase in the available throughput. For example, the evolution to Gigabit Ethernet and beyond, or the increase from single Mbps to double-digit Gbps from 3G to 5G mobile technologies. Nevertheless, corresponding reductions in latency have not kept pace [38, 80, 100], making latency the limiting factor for the deployment of new applications such as CPSs or the Tactile Internet. While a predictable Flow Completion Time (FCT) is crucial for real-time applications, it has also been demonstrated to impact the user experiences in less time-sensitive applications such as web browsing or file downloads [82, 100]. Even in these soft real-time applications that do not impose strict timing constraints, having predictably low delays is highly beneficial. TCP is the most widely deployed transport protocol and consequently, the FCT minimization problem has drawn significant attention from the TCP community over the last decade [64, 82, 91, 93, 112]. The rise of data-intensive applications such as machine learning has pushed the FCT down to the baseline fabric latency [108, 158], thereby approaching the theoretical minimum FCT of a fully reliable protocol such as TCP.

One line of research has focused on minimizing queueing delays via more efficient congestion control. Congestion control is the main mechanism behind TCP's performance,

as it regulates how much and when data is sent into the channel. Most CCAs use packet loss due to buffer overflow as the congestion signal—e.g., NewReno [87] or CUBIC [53]. However, these loss-based CCAs often suffer from *bufferbloat*: network buffers are kept full, thereby causing a queueing delay that is in the order of tens or hundreds of milliseconds for most packets [84], although it may be as large as several seconds in extreme cases [66, 75, 84]. Various CCAs, including DCTPC [64], PCC [102], TIMELY [108], BBR [112, 201], Copa [134] or HPCC [158] have emerged to address this issue by directly aiming to eliminate queues, thereby achieving lower and more predictable latency. Some of these algorithms [64, 201] leverage network support with Explicit Congestion Notification (ECN) [31, 175], which marks packets that experienced congestion for faster and more precise congestion detection. Precise in-network telemetry may also be used for more precise reactions to channel changes [158]. An alternative approach involves making congestion control aware of timing requirements [82, 91, 113]. For instance, per-flow timing information is distributed in [82] so that routers can reserve resources to meet the deadline of as many flows as possible. Vamanan et al. [91] leverage timing information in the congestion avoidance phase such that flows with a closer deadline get a larger proportion of the bottleneck data rate. A deadline-aware NewReno implementation is proposed in [113] which favors flows with a closer deadline to get a throughput above their fair share.

A second research direction has focused on enhancing loss recovery mechanisms in TCP to enable faster responses to packet losses. TCP's fully reliable, in-order delivery causes *head-of-line blocking* when packets are lost: out-of-sequence packets must be buffered at the receiver until losses are recovered before they can be forwarded to the application. Fast Retransmit [27] was originally proposed to trigger retransmissions upon the arrival of duplicated ACKs, instead of solely relying on slow timeouts. However, Fast Retransmit fails to operate effectively at the end of a transmission burst when no further packets are transmitted, which may trigger the duplicate ACKs mechanism. This problem, known as the *tail loss recovery problem* [93, 109], may increase the FCT by several orders of magnitude. In such cases, the flow experiences long tail latency, i.e., a few packets experience a delay significantly larger than the average packet delay, resulting in a characteristic tail-shaped probability density function as it approaches the x-axis. Flach et al. [93] have shown that proactively retransmitting packets without duplicate ACKs or timeouts reduces the tail latency and recent loss recovery standards already include this behavior by default [187]. Other standards propose better timeout management that adapts the waiting time to network conditions [79], thereby reducing the loss recovery time by one RTT [109]. Moreover, FEC has been introduced in TCP to recover losses without retransmissions [63, 93]. Sundararajan et al. [63] show that FEC significantly improves TCP's achievable throughput over lossy links, whereas Flach et al. [93] transmit a forward-encoded packet at the end of a transmission burst to mitigate tail latency.

As a result of the aforementioned performance improvements, TCP has become the prevalent protocol even in some time-sensitive niches—e.g., see Dynamic Adaptive Streaming over HTTP (DASH) [81] or datacenters [64, 91]. Nevertheless, as a fully

reliable protocol, TCP has some inherent theoretical limits to the timing guarantees it can provide.

2.3.3 Theoretical Limits

Shannon's channel coding theorem [13] states that the channel capacity C is defined in terms of the mutual information $I(x, y)$ as

$$C = \underset{p_x}{max}(I(x, y))$$

and fulfills the following property: for any $\epsilon > 0$ and $R < C$, for large enough N, there exists a code of length N and rate $r \geq R$ such that the probability of block error is $p_e \leq \epsilon$. In the limit—i.e., *large enough* N—, the number of symbol losses is close to the expected value. When operating in the finite block length regime, the number of losses deviates from the expected value, which introduces two inefficiency sources:

1. Fewer losses than expected occurred. More redundancy information is transmitted than required. The code operates below the channel capacity.

2. More losses than expected occurred. Less redundancy was transmitted than finally required, and thus $p_e > \epsilon$.

These inefficiencies prevent codes operating in the finite block length regime from reaching the optimal operation point. Polyanski et al. [70] derived a hard limit to how well these codes may approach the channel capacity.

Increasing data rates and the ability to directly operate with symbols (instead of packets) has enabled capacity-approaching codes in the lower layers of the stack—see Turbo [23], LT [32], or LDPC [15] codes. However, a different approach has been adopted in the packetized transport layer. The most widely deployed protocols, namely TCP and Quick UDP Internet Connections (QUIC), implement an ARQ-based error control mechanism. Thanks to feedback from the receiver, the protocol has precise knowledge of what data must be re-transmitted [24, 187]. ARQ approaches the channel capacity at the cost of some redundancy in the back channel in the form of acknowledgments. TCP does not have dedicated ACK messages but it incorporates this information in the headers of the packets sent by the other end in bidirectional communication, or in an empty packet if the communication is unidirectional. Throughput-intensive applications tend to utilize the complete MTU allowed by the physical channel, which in Ethernet-dominated networks such as the Internet typically is $1,500$ bytes. In contrast, the header field for the acknowledgments is 4 bytes long, thereby introducing a negligible redundancy overhead.

Shannon's coding theorem is completely time-agnostic. However, when strict time constraints are introduced, the ability of a code to achieve the desired reliability level depends on the time taken by the symbols to traverse the channel. Every packet transmission has an associated erasure probability, but the time constraint may prevent new retransmission cycles in ARQ or the transmission of enough parity packets in FEC. In

other words, there is a limit to the reliability level that can be achieved in time-constraint communications. A paradigm shift is required that puts the focus on predictable reliability instead.

2.3.4 Predictable Reliability

As long as transport protocols maintain full reliability as a core principle, time predictability may be improved, but a fully predictable delay remains impossible. Loss-triggered redundancy has been instrumental to the design of capacity-approaching error control in fully reliable protocols due to its low redundancy overhead. Nevertheless, when the deadline is too close to trigger any retransmission, less efficient proactive redundancy must be sent to reach the desired reliability level. To provide broad support for applications demanding a lower and more predictable delay [54, 101] and accomplish a sustainable deployment [34, 55, 216, 199], a transport protocol should be designed that:

1. Enforces a maximum tolerable delay, dictated by the application, between accepting data from the application and delivering it to the receiving end.

2. Provides predictable reliability within that time window, maintaining a packet loss rate that allows the application to operate safely.

3. Approaches the channel capacity by minimizing the transmitted redundancy so that it is close to the optimal minimum required to fulfill the timing and reliability requirements mentioned above.

The delay profiles of ARQ and FEC differ in nature. While the ARQ delay depends on the round-trip time between retransmission cycles, the FEC delay depends on the time required to collect symbols before encoding, typically in the form of packets in the transport layer—i.e., the inter-packet time. Therefore, the three design principles mentioned above aim to strike the optimal balance between FEC and ARQ, minimizing the required RI to achieve a target delay and packet loss rate.

The Real-time Transport Protocol (RTP) [36], originally designed for real-time media streaming, already implements some of the mechanisms to provide such a service. These include jitter compensation, timestamps for clock synchronization, sequence numbers for loss detection, and out-of-order delivery. RTP can also use FEC to correct losses without incurring an RTT [50] and the GCC [121] congestion control algorithm has been proposed to reduce queueing delay. The Datagram Congestion Control Protocol (DCCP) [44] is a straightforward mechanism to add congestion control to UDP for real-time applications. The Secure Reliable Transport (SRT) protocol [182] offers a time-aware delivery service that ignores late packets, implements congestion control, and combines FEC and ARQ for error control. Unreliable QUIC streams [212] have been proven to outperform TCP for the delivery of Video-on-Demand [144]. Zhang et al. [215] have made QUIC time-aware and added an FEC scheme to avoid slow retransmissions, thereby improving the QoE in DASH. The Loss-tolerant Transmission Protocol (LTP) [218] has been in-

tegrated into PyTorch to speed up the training of large, distributed Machine Learning models. Nevertheless, none of the aforementioned protocols combines time awareness with a predictably reliable, capacity-approaching error control scheme that minimizes the transmitted redundancy. The PRRT protocol [86, 163] developed at the Telecommunications Lab in Saarland University has been designed to fill that precise gap.

2.4 Conclusion

CPSs bridge the gap between the physical and the digital world by closing real-time control loops over embedded computers and networks. Given the broad set of application domains, these systems have the potential to permeate our day-to-day lives and dwarf the digital revolution we have experienced in the last decades. Thanks to the more rational use of resources and the real-time monitoring and control of processes, CPSs are a promising tool to face one of the major societal challenges of this century, namely climate change. However, current computing abstractions do not guarantee reliability and timeliness, the two functional requirements for correctness in CPSs. Additionally, energy efficiency should be considered as a non-functional requirement to ensure a sustainable deployment, so that the energy demand of these systems is only reduced as long as it has no negative impact on their correct operation. We envision open-source operating systems with open protocol stacks to play a similar role to the broad adoption of CPSs as they did for web-based applications. In particular, this thesis focuses on the transport layer, which is central to the communication process of distributed CPSs because it is their direct access to the communication channel. The end-to-end perspective of the communication channel allows the transport layer to ensure that the data is forwarded to the application within a specified delay budget while ensuring a target packet loss rate. However, widely deployed transport protocols have some foundational shortcomings that prevent them from fulfilling CPS requirements. The following chapter presents the PRRT protocol, which addresses these shortcomings, thereby enabling CPSs on a wide scale.

Chapter 3

Predictably Reliable, Real-Time Transport

Current standardization efforts aimed at meeting the predictable reliability and timing CPSs demand are restricted to deterministic performance guarantees in layers 2 with Time-Sensitive Networking (TSN) [139] and 3 with Deterministic Networking (DetNet) [149]. However, this determinism is not extended to the fourth layer, namely the transport layer, where commonly used protocols such as TCP, UDP, QUIC, RTP, and DCCP do not offer similar performance abstractions to the upper levels of the protocol stack. Consequently, applications lack a straightforward interface to the deterministic channels.

The fifth generation of mobile communication, also known as 5G, includes Ultra-Reliable and Low-Latency Communication (URLLC) as a target application. 5G-TSN deployments [143, 152, 225] have become common to support wireless CPSs such as swarm robotics [128, 159, 185, 226], or connected vehicles [2, 4, 78, 110]. Nevertheless, providing the same performance guarantees in the wireless scenario is a much more challenging task due to the dynamic behavior of the channel, which has been corroborated both in simulations [152] and real deployments [184]. There is a clear need for a transport protocol that not only provides performance guarantees to the application in terms of reliability and timeliness, but that is able to adapt to changes in the underlying channel to ensure the performance with a minimum resource footprint. The Predictably Reliable Real-time Transport (PRRT) protocol has been specifically designed to address this gap. This chapter presents the rationale behind the PRRT protocol and introduces the fundamental components that enable it to support CPSs.

3.1 Design Principles

The PRRT is the first of its kind to enable CPSs through a unique combination of cross-layer pacing and adaptive HARQ, which allows it to transmit data with the desired performance assurances—or alternatively, inform the application when the channel does not allow it—while minimizing its resource consumption.

3.1.1 Supported Applications

The target applications of a protocol greatly influence the set of available tools and techniques available for its design. While originally designed for real-time multimedia applications [86], PRRT has been later extended to accommodate CPSs as well [163]. This extension is justified by the shared characteristics of both application types, particularly in terms of latency and reliability requirements.

Real-time multimedia streaming applications demand a relatively constant data arrival rate to ensure seamless playback. Receiver buffers are employed to compensate for jitter, allowing for a certain reaction time before performance degradation becomes perceptible to the user. The size of the buffer is determined by the maximum tolerable delay for the specific application. For instance, a Video-on-Demand (VoD) service may employ buffers spanning several seconds [183], whereas video conference applications typically use buffers in the range of tens to hundreds of milliseconds [77]. In these applications, delayed packets are usually discarded without significant performance degradation due to error concealment—e.g., interpolating missing samples from neighboring ones—and even if they produce visible artifacts, often they are recovered after a few frames and they do not significantly impact the Quality-of-Experience (QoE) perceived by the user [107]. In summary, real-time multimedia applications are loss-tolerant, as no notable QoE drop appears as long as losses are maintained within a predictable threshold, but demand the data to be available before the hard deadline of the playback time.

Similarly, CPS data must be delivered with time and reliability assurances to ensure safe operation. However, CPSs introduce a nuance in the timing aspect that is frequently overlooked: although there are CPSs that do demand low latency [101], their main concern lies in time predictability so that messages are delivered and actions occur precisely when required [54]. It is important to note that not all the data must be received before the deadline as CPSs are weakly hard real-time systems that accept deadline misses so long as they occur in a predictable manner [30, 124, 196].

3.1.2 Self-Aware Transport

For PRRT to fulfill the reliability and timeliness requirements of the aforementioned applications, it must first implement an Application Programming Interface (API) for the CPSs to state its requirements. Once they are known, the protocol must be aware of the channel it operates on, as well as self-induced delays and packet losses that would also contribute to these targets. In other words, it must be a self-aware protocol as defined in [104], for which it must fulfill the following three properties:

1. **Self-reflective**: aware of its software architecture, execution environment, and hardware infrastructure on which it is running, as well as its operational goals.
2. **Self-predictive**: able to predict the effect of dynamic changes and the effect of possible adaptation actions.
3. **Self-adaptive**: proactively adapting as the environment evolves to ensure its operational goals are continuously met.

	TCP	UDP	QUIC	RTP	PRRT
Reliability	Full	None	Full/Partial	Partial	Predictable
Timeliness	None	None	None	Real-Time	Predictable
Congestion control	Yes	No	Yes	Yes	Yes
Error Control	ARQ	None	ARQ/FEC	FEC	HARQ
Self-Awareness	No	No	No	Partial	Yes
Security	TLS	None	TLS	SRTP	DTLS

Table 3.1: Qualitative comparison of the most common transport protocols with PRRT.

The PRRT protocol fulfills these three properties. First, it provides an API for the application to state its target packet loss rate for reliability and delay for timeliness (i.e., self-reflective). Second, in order to meet these two requirements, it measures processing and network delays (i.e., self-reflective) and builds a channel model to predict the expected packet loss rate and end-to-end delay (i.e., self-predictive). Third, the model is continuously updated so that the protocol can adapt its configuration under dynamic channel conditions (i.e., self-adaptive). Table 3.1 provides a qualitative comparison of PRRT with the most frequently used transport protocols, showing that only PRRT provides the desired characteristics to support CPSs.

3.1.3 Protocol Architecture

Fig. 3.1 illustrates the different components of the PRRT protocol. Time awareness is achieved by adding a timestamp and a timeout to every transmitted data packet. At the receiving end, only in-time packets are forwarded to the application using either the *asap*—i.e., as soon as possible—or *in-order* reception modes. On the one hand, the asap mode forwards every in-time packet as soon as they are received, and hence the application must tolerate packet reordering. On the other hand, in the in-order mode, the protocol waits for a reordering window upon the reception of every packet, in which out-of-sequence packets arriving in time may be reordered.

In order to fulfill the application constraints, the protocol must first be aware of them. PRRT provides an API for the application to state:

- **Target delay**: the maximum time that can elapse between data being accepted by the protocol and it being forwarded to the application on the receiver.
- **Target packet loss rate**: percentage of packets that can be lost at maximum by the protocol.

Once these constraints are known, the protocol leverages its tailored congestion and error control functions to ensure their compliance. Cross-Layer Pacing (X-Pace) extends congestion control to other layers of the protocol stack to avoid self-inflicted queueing delay not only in network buffers but in every buffer in the end-to-end pipeline. Adaptive HARQ finds the optimal balance between FEC and ARQ, such that the redundancy is minimized while ensuring that losses are corrected in time.

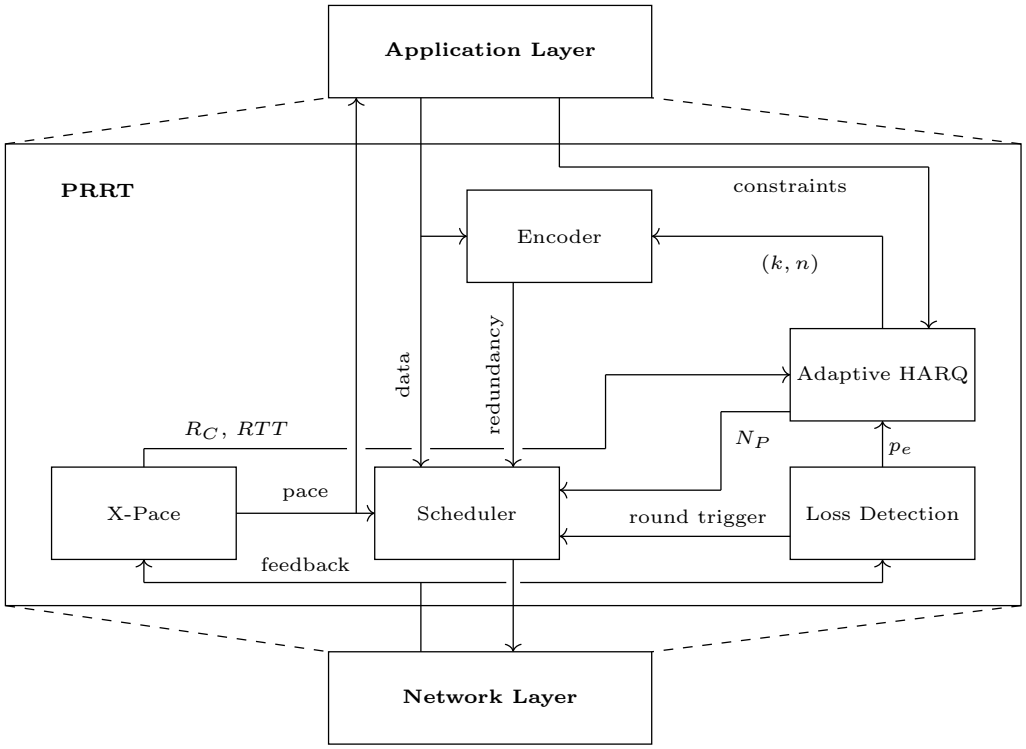
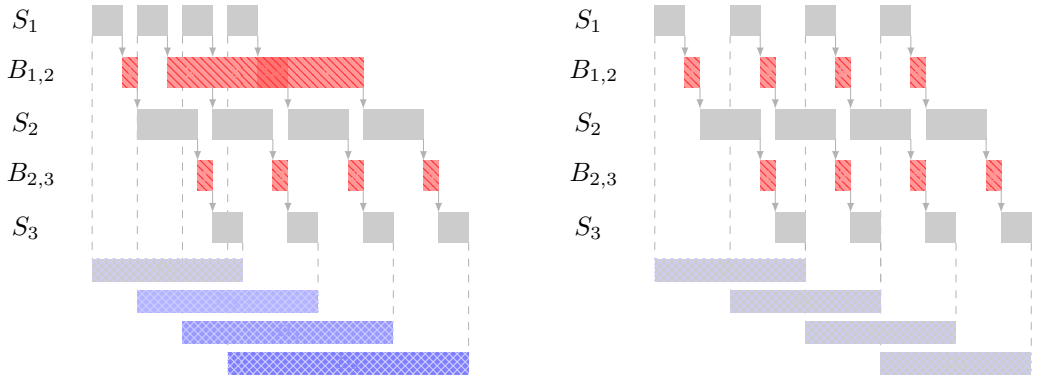


Figure 3.1: PRRT Architecture

3.2 Cross-Layer Pacing

Buffers have become an essential component in modern systems, facilitating asynchronous process-to-process communication and preventing data loss during sporadic bursts. However, a side effect of buffers is the emergence of bufferbloat whenever the arrival rate is consistently above the consumption rate [74, 84], thereby introducing a queueing delay. This issue is exemplified in Fig. 3.2a, which represents a system with three data processing steps (S_1, S_2, and S_3) and the buffers between these steps ($B_{1,2}$ and $B_{2,3}$). It can be observed that, as S_1 is not aware of the bottleneck of the system—i.e., the step that takes the longest to process a data unit, in this case S_2—, the packets it transmits are queued in $B_{1,2}$ until they can be processed. The four blue bars at the bottom represent the time elapsed since the creation of the data by the application, which increases as packets spend more time in $B_{1,2}$.

In order to tackle bufferbloat, pacing has been introduced in the transport layer [102, 112, 201] so that, instead of transmitting packets in bursts, they are spaced to achieve the desired data rate. In Fig. 3.2b pacing is applied to the previous example. S_1 is now aware of how long S_2 takes to process a data unit. Packets are consequently delayed—i.e., paced—so that they arrive at $B_{1,2}$ at the precise point in time that ensures the buffering time is minimum. As a result, packets experience a lower and more predictable delay when they arrive at S_3.



(a) Unpaced **(b)** Paced

Figure 3.2: Pacing keeps the bottleneck throughput but avoids excessive buffering latencies (marked as red striped) and increased E2E latencies (marked as blue hatched) [3]

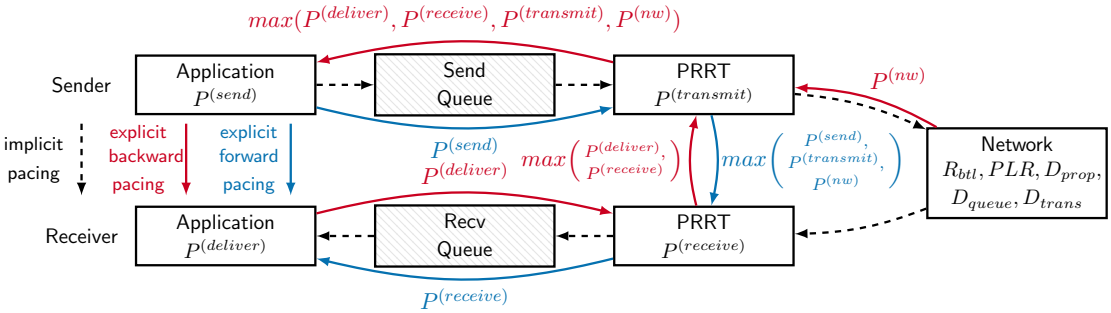


Figure 3.3: X-Pace system architecture. X-Pace measures and communicates the pace of the different steps, and adapts the paces across the entire system [3]

Buffers are present in several steps in the end-to-end pipeline seen by the application—e.g., in the transport layer, network cards, routers, etc. Therefore, in order to minimize the end-to-end delay, the pacing concept must be extended to more layers of the protocol stack. PRRT does so with X-Pace [3, 6, 163] to avoid the creation of permanent queues, regardless of where the buffers are.

Fig. 3.3 provides an overview of how X-Pace works within the context of PRRT. The protocol measures the pace—i.e., how long the system takes to process a data unit, for example, a packet—of the different steps. The sender and receiver application paces can be either directly provided by the application via an API or measured by PRRT. The protocol also measures the pace for the transmit and receive steps, as well as the network pace. X-Pace determines how long a packet takes to traverse the network with the bottleneck data rate and round-trip time estimators implemented in the BBR congestion control [203]. Once the protocol gathers the pacing information from all steps, it can determine where the bottleneck of the system—i.e., the step with the highest pace—is. Every step before this bottleneck must adapt to the bottleneck pace,

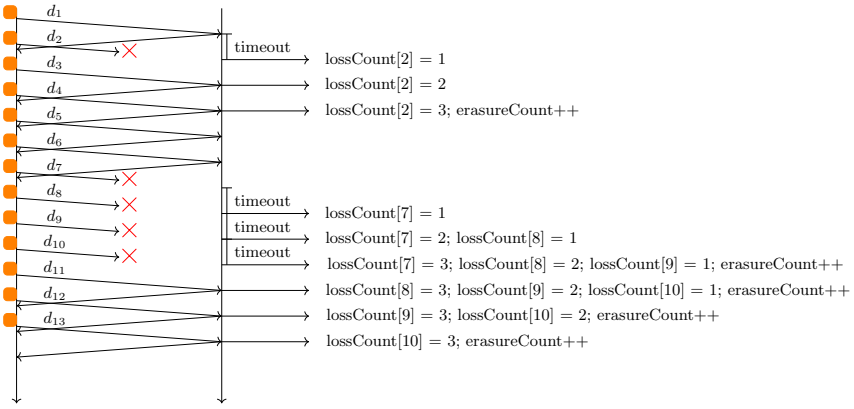


Figure 3.4: Loss detection algorithm with $loss\_threshold = 3$ and the timeout set to one inter-packet time ($D_{PL} = 3 \cdot T_s$). No packet reordering is considered.

or else queues would appear, whereas the steps behind the bottleneck may decide to adapt to the bottleneck—e.g., to reduce their resource footprint [5]—, but they are not strictly required to do so.

3.3 Adaptive HARQ

Error control in PRRT is based on HARQ, and its main components are depicted in Fig. 3.1. The HARQ-based error correction function employs an *encoder* to generate parity packets for loss recovery. *Loss detection* triggers reactive retransmission rounds, and a *scheduler* controls the number of parity packets transmitted in each round. Orchestrating the error control process is an *Adaptive HARQ* function that senses the channel to estimate its characteristics and configures the whole system to fulfill the application's target delay and packet loss rate constraints.

3.3.1 Loss Detection

Error control requires a mechanism to detect packet losses to either trigger repair cycles in ARQ or HARQ, or obtain the correct generator matrix based on the packets that have been received in FEC or HARQ. In PRRT, packets always carry a sequence number that is monotonically increased with the transmission of every new packet, and packet reception is acknowledged with positive ACKs.

The loss detection algorithm proposed in [68] operates on the receiver side, as illustrated in Fig. 3.4. A *lossCount* is maintained for every outstanding packet, which is increased upon two events: i) an out-of-sequence packet arrives, or ii) a timeout occurs before the packet's arrival. The timeout can take any value between one and two inter-packet times—i.e., $\lambda \in [T_s, 2 \cdot T_s]$. Once the counter reaches a specific threshold ($lossCount \geq loss\_threshold$), the packet is considered lost and a global erasure counter (*erasureCount*) is increased and notified to the sender (see Sec. 3.4). The algo-

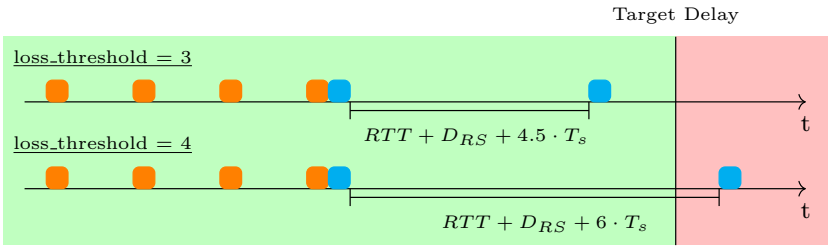


Figure 3.5: Impact of the *loss_threshold* parameter in loss detection on the ARQ delay. The sender must wait for losses to be detected before triggering a new round. In the worst case, the sender must wait for *loss_threshold* timeouts. The figure considers a timeout $\lambda = 1.5 \cdot T_s$ and $loss\_threshold = [3, 4]$.

rithm's robustness towards misclassifications caused by packet reordering is determined by *loss_threshold*, which also affects the loss detection delay D_{PL}. On the one hand, out-of-sequence packets or if $\lambda = T_s$ produce the fastest loss detection, in which case $D_{PL} \geq loss\_threshold \cdot T_s$. On the other hand, if $\lambda = 2 \cdot T_s$, loss detection can take at most *loss_threshold* timeouts to be detected, and hence $D_{PL} \leq 2 \cdot loss\_threshold \cdot T_s$. The timeout in PRRT is configured to $\lambda = 1.5 \cdot T_s$ to avoid waiting long times in the presence of long bursts of lost packets.

Under delay constraints, delayed packets arriving beyond the time budget are equivalent to lost packets. In addition, an excessive delay in loss detection may trigger retransmission cycles too late to meet the constraints, thereby forcing more parity packets into earlier cycles (see Fig. 3.5). Therefore, in the trade-off between loss miss-classification due to packet reordering and loss detection delay, PRRT is configured with a low threshold ($loss\_threshold = 3$) that avoids waiting a long time to trigger retransmission cycles. Such a threshold is in line with TCP's Fast Retransmit mechanism [27], which uses three duplicated ACKs as the signal to retransmit a packet.

The loss detection algorithm presented so far provides quick detection so long as the application has a relatively constant inter-packet time, which is the case for real-time applications and CPSs. However, the transmission of parity packets occurs at a different rate that depends on the RTT between retransmission cycles instead of the T_s as the data packets. If $T_s \ll RTT$, timeouts expire faster than the sender transmits parity packets. Conversely, if $RTT \ll T_s$, parity packet loss detection takes significantly longer than the parity packet arrival time. The latter scenario is akin to the tail loss recovery problem encountered in TCP [93, 109], where the absence of duplicated ACKs at the end of a transmission burst results in slow error recovery. The tail loss recovery problem has been solved by taking proactive measures at the end of the burst to avoid waiting for timeouts [93, 187]. PRRT addresses this issue by implementing *Preemptive Loss Notifications*. Upon receiving the first parity packet for a block, the receiver expects the subsequent parity packet to arrive within the next Δ_{rr} milliseconds, defined as the time between retransmission cycles (see Eq. (3.1)). If no parity packet is received within this timeframe, the receiver sends a preemptive Negative Acknowledgment (NAK), allowing

parity packet loss recovery within one RTT. The sender treats the NAK as a loss if it had already transmitted the corresponding parity packet; otherwise, the NAK is ignored.

3.3.2 Scheduler

The *scheduler* determines how parity packets are injected into the stream of data packets. The parameter N_P governs whether they are transmitted in the proactive FEC cycle or one of the N_C reactive ARQ cycles, where every entry $N_P[i]$ is the number of parity packets transmitted in the i'th cycle. $i = 0$ corresponds to FEC and is always scheduled immediately after transmitting the k'th packet in the block. On the other hand, time-triggered scheduling is used for ARQ cycles: the sender must wait for at least one RTT for the ACK of the last transmitted packet, and at most $RTT + D_{PL}$ to account for the loss detection below (see the loss detection algorithm above). The term D_{RS} is added to account for the packet processing time at the receiver and ACK processing time at the sender, thereby avoiding retransmission cycles being triggered too early.

$$\Delta_{rr} = RTT + D_{PL} + D_{RS} \tag{3.1}$$

For every block, a timer is set to trigger a new retransmission cycle after Δ_{rr} milliseconds (see Eq. (3.1)) if a loss has been detected. If no loss is detected, the cycle is skipped and a new timer is set. This process continues until either the receiver signals that k packets have been received or the sender exhausts all the available rounds for the block.

3.3.3 En-/Decoder

PRRT's *encoder* and *decoder* use systematic MDS Vandermonde code in the Galois Field $GF(2^8)$ and hence the en-/decoding operations are performed byte-wise. This component is fully configurable, i.e., PRRT can use any $\mathcal{C}(n, k)$ code with $k, p \in [0, 255]$; $s.t.\ n \leq 255$.[1] The two basic coding operations, namely matrix-vector multiplication and matrix inversion, have been implemented with dynamic programming to avoid costly modulo operations [26]. Given a non-zero field element $x = \alpha^{k_x}$ (here k_x can be considered the logarithm of x), field multiplication and division can be computed with Eq. (3.2) and Eq. (3.3), respectively.

$$x \cdot y = \alpha^{|k_x + k_y|_{q-1}} \tag{3.2}$$

$$\frac{1}{x} = \alpha^{q-1-k_x} \tag{3.3}$$

PRRT pre-computes four tables offline, consisting of i) every field in the element indexed by their exponent, ii) the logarithm of every field element, iii) the division of

[1] In systematic form, a code in $GF(2^8)$ allows for $n \leq 510$. However, we have limited $n \in [0, 255]$ to reduce the solution search space (see Sec. 4.2), where $n = 0$ signals that no valid configuration is available that meets the constraints (more on this in Sec. 3.5.1).

every field element, and iv) the multiplication of every two field elements. The size of these tables depends on the field size $q = 2^m$. They have a memory complexity $\mathcal{O}(2^m)$ except for the multiplication table, which has a complexity $\mathcal{O}(2^{2m})$.

Multiplication and division operations boil down to a table lookup. The complexity for the matrix-vector multiplication is $\mathcal{O}(ke)$ and $\mathcal{O}(ke^2)$ for the matrix inversion, where $e \leq min(k, n-k)$ the number of lost packets. The Gauss-Jordan algorithm is used to perform the matrix inversion, which benefits from the code being systematic because row reduction must only be used in the e columns stemming from parity packets. Fig. 3.6 depicts how the matrix-vector multiplication is performed over complete packets. Given that symbols are one byte long, the operation must be iterated throughout the complete packet length (L_p), both for encoding and decoding, whereas the matrix inversion must only be done once per decoding operation.

$$\mathcal{C}_{mul}^{MDS} = \mathcal{O}(keL_p) \tag{3.4}$$

$$\mathcal{C}_{inv}^{MDS} = \mathcal{O}(ke^2) \tag{3.5}$$

In Ethernet deployments, $L_p = MTU = 1,500$ bytes. Therefore, $L_p \gg e$—i.e., at maximum p packets can be lost and $p \leq 255$ in $GF(2^8)$—, and hence the complexity of the multiplication ($\mathcal{C}_{mul}$) dominates over the complexity of the inversion ($\mathcal{C}_{inv}$). A higher-order field would increase the symbol length and shorten the iteration throughout the complete packet. However, this complexity reduction approach would turn PRRT unable to run on constrained devices because the size of the dynamic tables mentioned above grows exponentially with the number of bits per symbol. Chapter 6 addresses this issue more in detail and proposes an alternative, energy-efficient coding function for PRRT.

3.3.4 Channel Estimation

For the adaptive HARQ function to adapt its performance to the underlying channel, it receives estimates for the data rate (R_C), round-trip time (RTT) and erasure rate (p_e) from the X-Pace and Loss Detection components (see Fig. 3.1. X-Pace implements BBR's channel estimation [112, 203] with the data rate and RTT estimations in Eqs. (3.6) and (3.7), respectively.

$$\widehat{R_C} = max(\boldsymbol{R_C}[t]) \;\; \forall\, t \in [T - W_B, T] \tag{3.6}$$

$$\widehat{RTT} = min(\boldsymbol{RTT}[t]) \;\; \forall\, t \in [T - W_R, T] \tag{3.7}$$

The vectors $\boldsymbol{R_C}$ and $\boldsymbol{RTT}$ contain samples for each metric obtained in the time window W_B and W_R, respectively. The algorithm in [203] samples the data rate and round-trip time at two different granularities: the time window W_B is typically six to ten RTTs, while the window W_R is typically tens of seconds to minutes. These window

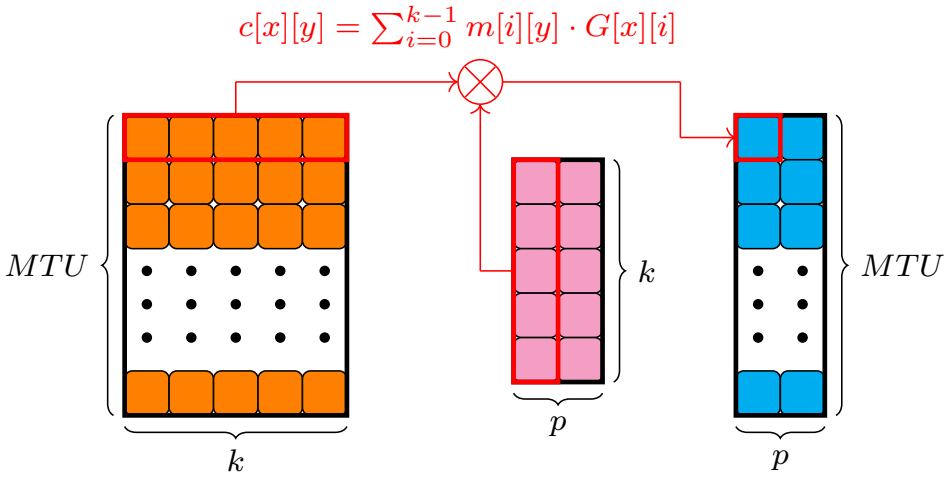


Figure 3.6: Visualization of the matrix-vector multiplication performed by the PRRT protocol. Given the message vector $\mathbf{m}$, the parity vector $\mathbf{c}$ is $\mathbf{c} = \mathbf{m} \cdot \mathbf{G}$, where $\mathbf{G}$ is the generator matrix of a systematic code. In a packetized layer with virtual interleaving, $\mathbf{m}$ and $\mathbf{c}$ are matrices of size $k \times MTU$ and $p \times MTU$, respectively. Therefore, the operation must be iterated throughout the complete packet length to encode a complete packet. Each square in the picture represents an element in $GF(2^8)$—i.e., a byte.

values assume fast-changing data rates due to congestion while the RTT is relatively stable for longer periods because path changes occur on larger time scales [112].

PRRT's Loss Detection function keeps track of whether a packet is marked as received or lost in the ***loss*** vector, in which a 0 denotes a received packet and a 1 means the packet was lost. The channel erasure rate is estimated as the sample mean of the lost packets in a sequence of W_L packets (see Eq. (3.8)), which is the maximum-likelihood estimator for the p_e [86]. The window size can be configured to achieve a specific accuracy of $\pm a_{p_e}$ with a confidence level ν with Eq. (3.9). According to [86], $W_L = 16,500$ samples can estimate a channel erasure rate of 1% with an accuracy of ± 0.2 with 99% confidence level.

$$\widehat{p_e} = \frac{1}{W_L} \cdot \sum_{t=1}^{W_L} \boldsymbol{loss\_sequence}[t] \quad \forall\, t \in [T - W_L, T] \tag{3.8}$$

$$W_L = \left(\frac{1}{\widehat{p_e}} - 1\right) \cdot \left(\frac{\nu}{a_{p_e}}\right)^2 \tag{3.9}$$

This thesis considers the simple i.i.d. channel model, which does not model the temporal correlation of packet losses that are common in IP networks—e.g., if a network buffer overflows. [86] shows that the low precision of such a channel model can be compensated with more conservative decisions for the PRRT parameters. Therefore,

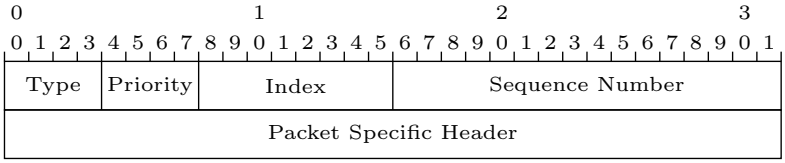


Figure 3.7: PRRT General Header

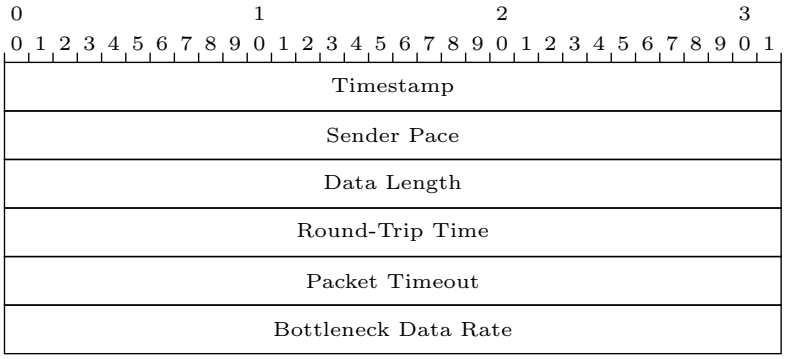


Figure 3.8: PRRT Data Header

the implementation of more sophisticated channel models, such as the Gilbert-Elliot model, is not justified in PRRT given their higher complexity.

3.4 Packet Format

Every PRRT packet begins with the *General Header* depicted in Fig. 3.7. This header begins with the packet *Type*—i.e., Data, Parity, Feedback, or NAK—, followed by the *Priority* field, which is currently unused. The *Index* field signals the position of the packet in the coding block it belongs to (see Sec. 2.3.1 for more details). The *Sequence Number* is independently treated for every packet type—i.e., the Sequence Number does not univocally identify a packet, but the (Type, Sequence Number) tuple does.

Every General Header is followed by a packet-specific header. The *Data Header* (Fig. 3.8) begins with the *Timestamp*, which represents the acceptance time of the data in the payload by the protocol stack. *Sender Pace* is the maximum pace at the sender side, which is followed by the length of the payload in bytes (*Data Length*). The estimated round-trip time and bottleneck data rate are communicated in the *Round-Trip Time* and *Bottleneck Data Rate* fields, respectively. The packet expiration time, calculated by adding the application target delay to the Timestamp, is put in tthe *Packet Timeout* field.

Parity packets carry all the necessary information to configure the decoder in its header, depicted in Fig. 3.9. As in the Data Header, the packet creation timestamp (*Timestamp*) and the sender pace (*Sender Pace*) are the first two fields in the header. The parity-specific fields are the codeword length (n), block length (k) and *Base Sequence*

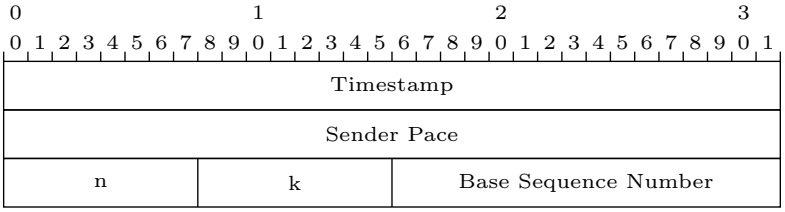


Figure 3.9: PRRT Parity Header

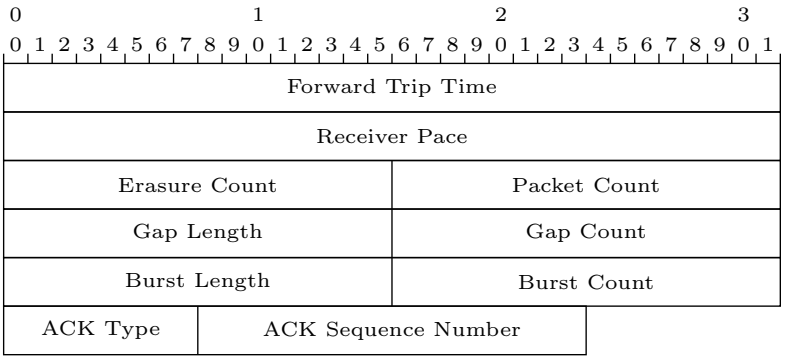


Figure 3.10: PRRT Feedback Header

Number fields. The latter is the sequence number of the first data packet in the block the parity packet belongs to, and is used by PRRT for block indexing.

Feedback packets notify timing information (*Forward Trip Time* and *Receiver Pace*), packet reception statistics (*Erasure Count* and *Packet Count*), information on bursts of lost (*Gap Length* and *Gap Count*) and received packets (*Burst Length* and *Burst Count*). In addition, the header (see Fig. 3.10) contains information to univocally identify the acknowledged packet (*ACK Type* and *ACK Sequence Number*).

NAKs are only used in PRRT to preemptively signal the loss of parity packets. Therefore, their header only contains the sequence number of the parity packet deemed as lost (*NAK Sequence Number*) and the base sequence number of the block it belongs to (*NAK Base Sequence Number*).

3.5 Optimal Hybrid Erasure Coding

For the adaptive HARQ function to adapt its performance, it does not only require the application constraints and a channel model but also the definition of a desired metric the system must be optimized against. Minimizing the transmitted redundancy is essential for any networked system because it reduces the resource footprint—i.e., fewer packets are encoded and transmitted, with the power draw it entails—, and is fair with other systems the network resources are shared with, one of the fundamental aspects in transport layer design [169]. The centrality of the RI motivates its selection as

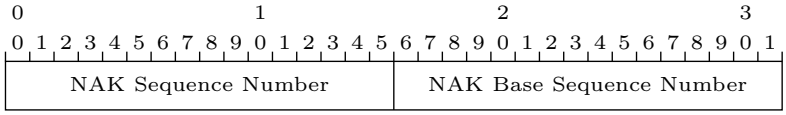


Figure 3.11: PRRT NAK Header

the function to minimize. In the following, a formal model of the optimization problem adaptive HARQ solves is presented.

3.5.1 The Optimization Problem

Every HARQ scheme is fully determined by two parameters: the block length (k) and the repair schedule (N_P), which in turn dictates the number of parity packets ($p = ||N_P||_1$, with $||N_P||_1$ the 1-norm of the repair schedule vector) and the number of retransmission cycles (N_C). Therefore, adaptive HARQ's task is finding the optimal (k, N_P) tuple that minimizes the RI. The optimization problem is formally defined in Eq. (3.10).

$$\begin{aligned} k^*, {N_P}^* = & \arg\min_{k,N_P} RI(k, N_C, N_P) \\ \textit{such that: } & D_{HARQ}(k, N_C, N_P) \leq D_T \\ & PLR_{HARQ}(k, \|N_P\|_1) \leq PLR_T \\ & Dr(k, N_C, N_P) \leq R_C \\ & k, p, N_C \in \mathbb{N} \\ & k, p, N_C \leq 255 \end{aligned} \tag{3.10}$$

The selected configuration must fulfill five constraints:

(i) The delay of any packet in the block ($D_{HARQ}(k, N_C, N_P)$) must be below the application's target delay (D_T).

(ii) The average packet loss rate ($PLR_{HARQ}(k, ||N_P||_1)$) must be below the application's target loss rate (PLR_T).

(iii) The transmitted data rate ($Dr(k, N_C, N_P)$) must be below the channel's bottleneck data rate to avoid congestion (R_C).

(iv) The variables k, p, and N_C are natural numbers because the coder can only operate on full symbols and a fraction of a repair cycle cannot exist.

(v) The employed MDS code forces $k, p \in [0, 255]$—the optimization algorithms outputs $k = 0$ only when no configuration is found that satisfies all the configurations at the same time. Since every repair cycle must transmit at least one parity packet, $N_C \leq 255$ also holds.

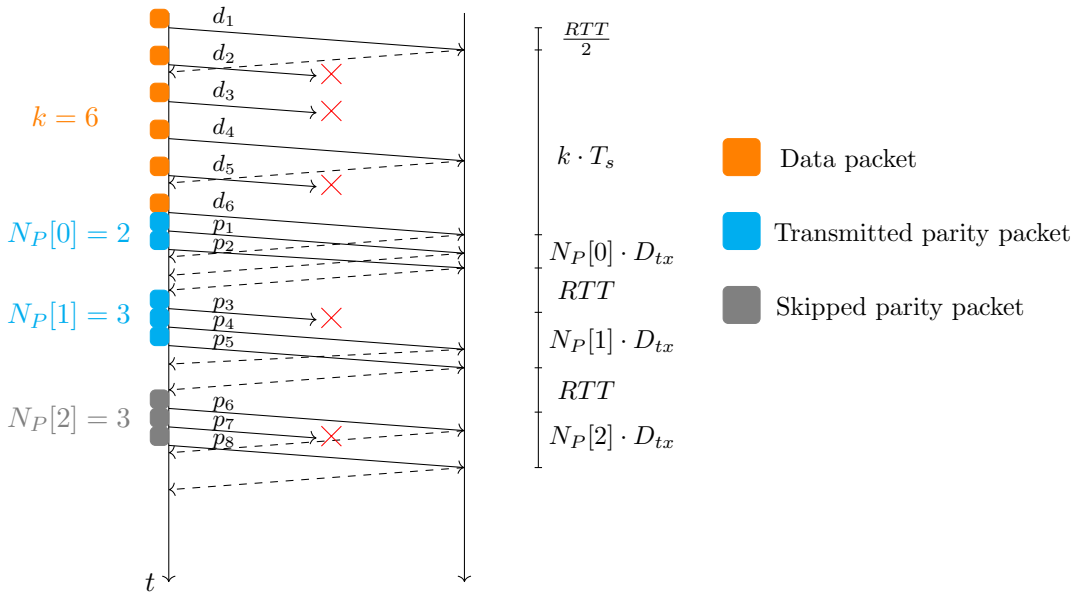


Figure 3.12: HARQ delay budget

Channel Model

The RI is defined in Eq. (3.11), which is no more than the weighted sum of the transmitted parity packets in each cycle. The weight $p_f^M[c]$ is the probability of each new cycle being triggered in a multicast group with M receivers—see Eq. (3.12). From the perspective of a single receiver, a new repair cycle must be triggered if less than k packets were received but no more packets were lost than can be recovered with the remaining parity packets in the schedule. In the latter case, the equation system remains underdetermined even in the event of receiving all the remaining parity packets, and hence their transmission is futile. For example, if $k = 10$ and $N_P = [0, 1, 1]$, if three losses are detected before triggering the first ARQ cycle, the repair cycles need not be scheduled. This repair cycle scheduling mechanism is modeled in Eq. (3.13). We assume the sender keeps triggering retransmission cycles until all receivers receive k packets.

$$RI(k, N_C, N_P) = \frac{1}{k} N_P[0] + \frac{1}{k} \sum_{c=1}^{N_C} p_f^M[c] \cdot N_P[c] \tag{3.11}$$

$$p_f^M[c] = 1 - (1 - p_f[c])^M \tag{3.12}$$

$$p_f[c] = \sum_{i=max(0,\mathbf{n}[c-1]-p)}^{k-1} \binom{\mathbf{n}[c-1]}{i} (1 - p_e)^i p_e{}^{\mathbf{n}[c-1]-i} \tag{3.13}$$

While the FEC delay Eq. (3.14) depends on the source packet interval T_s to collect k data packets before encoding, the ARQ delay Eq. (3.15) is RTT-dominated due to the

ACK-triggered retransmission process. The HARQ delay Eq. (3.16) can be represented as the combination of its FEC and ARQ components, as depicted in Fig. 3.12.[2] Retransmission cycles are only triggered upon the detection of packet losses (see Sec. 3.3.1), and thus the time the system takes to detect losses must be considered in the model. The loss detection algorithm introduced in Sec. 3.3.1 is configured with a timeout $\lambda = 1.5 \cdot T_s$ and a threshold $loss\_threshold = 3$, which results in a loss detection delay $D_{PL} = 4.5 \cdot T_s$. This configuration has been selected because it provides a good trade-off between loss misclassification and fast retransmission cycles. Finally, D_{RS} models the processing delay introduced by the protocol, which can be obtained from X-Pace (see Sec. 3.2).

$$D_{FEC}(k, N_P) = \frac{RTT + D_{RS}}{2} + k \cdot T_s + N_P[0] \cdot D_{tx} \tag{3.14}$$

$$\begin{aligned} D_{ARQ}(N_C, N_P) &= \frac{RTT}{2} + \sum_{c=1}^{N_C} (RTT + N_P[c] \cdot D_{tx} + D_{RS} + D_{PL}) \\ &= \frac{RTT}{2} + N_C \cdot (RTT + D_{RS} + D_{PL}) + (||N_P||_1 - N_P[0]) \cdot D_{tx} \end{aligned} \tag{3.15}$$

$$D_{HARQ}(k, N_C, N_P) = D_{FEC}(k, N_P) + D_{ARQ}(N_C, N_P) - \frac{RTT}{2} \tag{3.16}$$

The constant inter-packet time in Eq. (3.14) assumes some periodic behavior from the application—e.g., video streaming with a constant frame rate, sensors in CPS with a constant sampling rate, or periodic control messages as the controller reacts to those sensor samples. In some cases, these real-time applications react to changes in network conditions by adapting their data rate. For example, Adaptive Bit Rate (ABR) algorithms in DASH [126, 183] and adaptive sampling rate in networked control systems [99]. Nevertheless, the data rate is still governed by the application and this information could be provided via PRRT's API. On the contrary, the T_s estimation function must be able to detect bursts for PRRT to support bursty, time-aware traffic. PRRT can detect a burst when no more data is provided after T_s, in which case a new constraint is added that caps k to the maximum achievable block length for such a burst.

The packet loss rate is given in (3.17), where $P(I_k = i)$ is the probability of being unable to decode exactly i data packets—i.e., the loss rate as seen by the application—when a systematic MDS code is used. The expression considers binary erasure channels with i.i.d. losses.

$$PLR_{HARQ}(k, p) = \frac{1}{k} \sum_{i=1}^{k} i \cdot Pr(I_k = i) \tag{3.17}$$

[2]The delay models presented here assume symmetrical network delays for simplicity. However, the X-Pace function in PRRT provides precise timing information for both directions of the communication channel, and hence the protocol can be easily extended to channel with asymmetrical delays.

$$Pr(I_k = i) = \sum_{e=max(p+1,i)}^{p+i} \binom{n}{e} \cdot {p_e}^e \cdot (1-p_e)^{n-e} \cdot p_d\binom{e}{i} \tag{3.18}$$

$$p_d\binom{e}{i} = \frac{\binom{k}{i}\binom{n-k}{e-i}}{\binom{n}{e}} \tag{3.19}$$

While the previous two constraints deal purely with application constraints, the data rate constraint avoids network congestion by ensuring that the transmitted data rate (3.20) is below the bottleneck data rate of the network R_C.

$$Dr(k, N_C, N_P) = (1 + RI(k, N_C, N_P)) \cdot \frac{L_p}{T_s} \tag{3.20}$$

The three constraints are expected to be checked relatively often, since PRRT must check them before taking any (k, N_P) pair as a valid configuration. Therefore, reducing the run-time complexity of the constraints check becomes imperative. While the delay expression can be directly evaluated in $\mathcal{O}(1)$—see Eq. (3.16)—, a recursive approach has been used to reduce the complexity of the PLR and data rate evaluation. Appendix A presents an algorithm to obtain Eq. (3.17) with run-time complexity $\mathcal{O}(k + log(p))$, whereas Eq. (3.20) can be evaluated in $\mathcal{O}(k + p)$, as shown in Appendix B.

3.6 The need for HARQ

The formal definition of the time-aware error control problem provided in Eq. (3.10) enables us to test the fundamental hypothesis behind the PRRT protocol: *an HARQ-enabled error control function is essential for the efficient transport of time-aware data.* A dataset of 2.5 million samples resembling realistic network conditions has been created to test this hypothesis. The samples are input into three different versions of the optimization problem in Eq. (3.10): i) an HARQ version with the problem as it is, ii) a pure ARQ version, and iii) a pure FEC version. Pure ARQ is enforced by ensuring that no packet is encoded ($k \leq 1$), the proactive cycle cannot be used ($N_P[0] = 0$), and no proactive packets are transmitted in later rounds ($N_P[i] \leq 1 \; \forall \, i > 1$). On the other hand, pure FEC is enforced with the constraint $N_C = 0$ so that redundancy is solely transmitted in the proactive cycle.

3.6.1 Methodology

The design of a dataset that represents realistic network conditions is instrumental for the evaluation of the hypothesis. The dataset considers 6 input parameters:

- Application parameters: target delay (D_T), packet loss rate (PLR_T), and source packet interval (T_s).
- Network parameters: channel erasure rate (p_e), round-trip time (RTT), and bottleneck data rate (R_C).

Table 3.2: Selected orders of magnitude for the generation of the parameter dataset.

Parameter	Orders of Magnitude	Unit	Reference
PLR_T	10^{-3}, 10^{-4}, 10^{-5}	rate	[77]
D_T	10^{0}, 10^{1}, 10^{2}	ms	[77, 101, 125]
p_e	10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}	rate	[131, 184]
RTT	10^{0}, 10^{1}, 10^{2}	ms	[120, 184]
R_C	10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}	Mbps	[120, 126, 145, 184]
T_s	10^{-1},10^{0}, 10^{1}	ms	[94, 136]

Faithfulness is achieved by considering throughput, round-trip time, and loss rate traces collected in the wild for the most common network deployments—i.e., broadband [126], 4G [145], 5G [178, 184], and WiFi [120, 131]. When it comes to application parameters, the dataset considers the requirements the PRRT has been designed to support: real-time multimedia applications [77, 94] and CPSs [101, 125, 136]. For every parameter, Table 3.2 collects orders of magnitude that are frequently found in the wild, according to the aforementioned references. For the generation of the dataset, an order of magnitude in Table 3.2 is randomly selected for every parameter, together with a leading number between 1 and 9. This process is repeated 2.5 million times in order to achieve good generalizability.

The delay and loss rate models consider three other parameters that have not been included in the dataset here presented (see Eq. (3.16) and Eq. (3.17)). The packet length is assumed to be fixed (L_p) to the MTU of the underlying physical medium, which in IP networks is typically limited by the Ethernet MTU (i.e., $L_p = 1,500$ bytes). Although precise processing delay information could be fed into the algorithm [3, 6], a rather conservative constant processing delay $D_{RS} = 1$ ms has been considered for a device-agnostic analysis. Finally, given the loss detection mechanism implemented in PRRT (see Sec. 3.3), $D_{PL} = 4.5 \cdot T_s$, which is the upper bound to the loss detection delay.

Definition 3.6.1 (Valid Configuration)**.** An HARQ configuration (k, N_P) is a *valid configuration* if, given the channel model $\mathcal{M}(p_e, RTT, R_C)$ observed by the protocol, it fulfills the D_T, PLR_T and R_C constraints in Eq. (3.10).

Definition 3.6.2 (Optimal Configuration)**.** An HARQ configuration (k, N_P) is an *optimal configuration* if, given the channel model $\mathcal{M}(p_e, RTT, R_C)$ observed by the protocol, it fulfills the D_T, PLR_T and R_C constraints in Eq. (3.10) with minimum RI—i.e., it is the solution to the optimization problem in Eq. (3.10).

For the comparison of the error control schemes, it is important to differentiate between the concepts of a *valid configuration* and an *optimal configuration*, which are defined in Def. 3.6.1 and Def. 3.6.2, respectively. While the latter is a measure of optimality,[3] the former measures the ability of a scheme to provide broad support for CPSs

[3]It should be noted that optimality here refers to the optimal configuration for a fixed coding technique and not the overall optimal configuration for all available codes. The latter is a more complex optimization problem that is further discussed in Sec. 6.6.4.

Table 3.3: Total number and rate of valid and optimal configurations for every error control scheme. The results have been obtained after running Eq. (3.10) through the complete dataset. As HARQ supports any combination of the other two schemes, its number of valid configurations is taken as the reference of the maximum possible supported configurations. The rate normalized to the number of valid configurations for HARQ is presented in the *normalized* column.

Scheme	Valid Configurations			Optimal Configurations		
	Total	**Rate**	**Normalized**	**Total**	**Rate**	**Normalized**
ARQ	530,188	21.20%	49.70%	67,508	2.70%	6.32%
FEC	970,085	38.80%	90.95%	287,090	11.48%	26.91%
HARQ	1,066,628	42.66%	100%	1,066,628	42.66%	100%

on a wide range of network conditions. HARQ is taken as the reference for the other two schemes because it can configure any combination of proactive and reactive redundancy, including pure FEC and pure ARQ. Therefore, if HARQ finds no configuration that fulfills the constraints, neither will FEC nor ARQ.

3.6.2 Evaluation

Table 3.3 shows the total number and rate of supported configurations by every error control scheme. A scheme is considered to provide a valid configuration for an input if it is able to fulfill all the constraints of the optimization problem at the same time. HARQ clearly outperforms the other two schemes, supporting more than twice the number of cases as ARQ and 9.05% more cases than FEC. Although FEC's coverage of valid configurations is close to HARQ's, it produces a constant redundancy overhead regardless of the number of packet losses experienced in a transmission. Fig. 3.13 shows the Cumulative Distribution Function (CDF) of the RI obtained by every scheme. To enable a direct comparison among the schemes, the depicted CDFs show the results for the complete dataset—i.e., the remaining probabilities up to 1 correspond to the invalid configurations. Despite its low valid configuration coverage, ARQ achieves lower RI than FEC in a significant number of cases. Such a low overhead is most likely a result of the feedback mechanism, which enables precision loss recovery only when packets are known to be lost. On the contrary, FEC approaches the coverage of HARQ at the cost of large RI due to the more conservative decisions resulting from the lack of a feedback channel. Therefore, the remaining 3.86% that FEC does not support but HARQ does are probably cases in which the data rate exceeded the channel data rate (R_C) due to the large RI increase.

3.7 Discussion

The fully reliable paradigm followed by TCP fails to provide the timing guarantees that CPSs demand for a correct operation. This chapter has highlighted the need for a

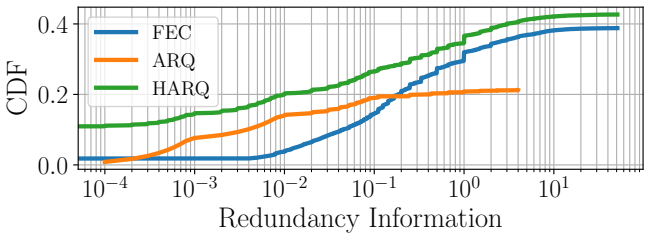


Figure 3.13: CDF of the RI increase in percentage for the ARQ and FEC schemes in comparison to the optimal achieved by HARQ.

double paradigm shift: i) from pure ARQ to adaptive HARQ to provide timely error recovery, and ii) from fully reliable, time-insensitive transport to predictably reliable, timely transport. A thorough discussion of this affirmation is provided in the following.

The need for HARQ

As shown in Sec. 3.6, pure ARQ and FEC fail to i) support time-aware transport in a broad number of network scenarios, ii) and provide efficient time-aware error control, respectively. On the one hand, the poor performance of a purely reactive scheme comes from a large amount of time spent on detecting packet losses (D_{PL}) and triggering retransmissions (RTT). On the other hand, purely proactive redundancy causes a fixed transmission of parity packets regardless of whether they are required. A fully adaptive HARQ scheme brings the best of both worlds: as the deadline approaches, proactive redundancy may be transmitted to reach the desired reliability level, while feedback can be obtained if time allows in order to avoid unnecessary transmissions. Moreover, HARQ still allows for a pure ARQ to be configured in those platforms for which packet coding is deemed computationally expensive. Despite the proposals to include it into TCP [41, 63], it has not been until the release of QUIC that HARQ has been deployed at a wide scale [144, 151, 160, 193, 198, 211]. Nevertheless, HARQ alone is not enough to support the stringent requirements of CPSs and a paradigm shift is required in transport protocol design in order to support applications with stringent timing requirements such as CPSs.

Paradigm Shift

Decades of research in TCP have reduced its delay down to the fabric delay due to improved congestion avoidance that prevents queues from being created at the switches and routers [108, 158]. Consequently, fully reliable transport has reached its theoretical limits to predictable delay. The head-of-line blocking and tail loss recovery problems, both increasing the delay by several RTTs, are a direct consequence of a fully reliable transport service. Therefore, a more predictable delay must be accompanied by a paradigm shift at the core of the transport layer [166] to provide a partially reliable service. Aware of this issue from early on, the networking community has proposed multiple protocols

that provide such a service [36, 44, 182, 212, 215]. Although all these protocols fail to provide predictable reliability within the application's delay constraint, most of the components that are required to do so have been developed around them throughout the years: MDS codes for packetized layers [26] to enable HARQ at the transport layer, time-awareness [36, 182] and a common sense of time in distributed deployments with the NTP [67] or PTP [156] protocols, transport protocols obtaining the application requirements via an API [223], and channel sensing in the transport layer [36, 112]. The original contribution of the PRRT protocol is the combination of these components to go one step ahead and provide predictable reliability instead. Similarly to other real-time protocols, PRRT avoids waiting for late packet retransmissions with an addition: the application specifies the maximum tolerable delay, and hence PRRT discards late packets even when they are not lost. Unlike other protocols, PRRT ensures the percentage of lost packets is never above a threshold that is also configurable by the application.

Predictable Reliability

PRRT's ability to fulfill the application constraints depends on i) how much its channel model resembles the actual communication channel, and ii) how quickly it adapts its performance to changes in the channel. Although channel sensing is out of the scope of this thesis, it is a prolific research topic on its own [47, 86, 146, 171, 191, 203], and future work should investigate whether PRRT could benefit from some of the recently proposed models. Long Short-Term Memory (LSTM) neural networks have shown promising results when applied to this problem [146, 171, 191], which has been shown to outperform non-ML solutions [146]. However, this thesis focuses on PRRT's ability to react to channel changes. While the optimization problem described in Sec. 3.5 is executed, PRRT does not have any guarantees regarding its ability to meet the performance levels requested by the application. In the worst case, the channel coherence time is shorter than PRRT's reaction time, thereby making it impossible to provide any guarantees at all. Chapter 4 shows the optimization problem is a computationally complex process, meaning that the aforementioned worst case is not actually too farfetched. Another important aspect to consider is the resource footprint of the system. Minimizing the transmitted RI does indeed optimize the protocol resource footprint but only in the network. Executing a computationally expensive algorithm to do so puts a heavier burden on the CPU that must be accounted for. All in all, there is a need for a computationally efficient adaptive HARQ function.

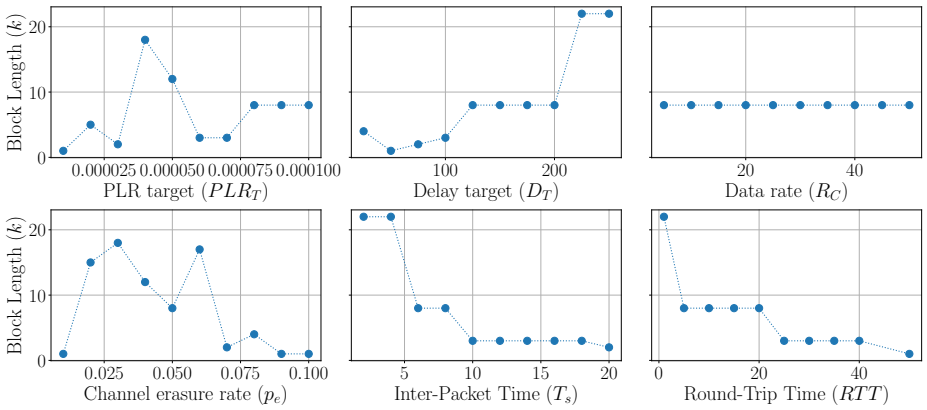
Chapter 4

Search Algorithms for Adaptive HARQ

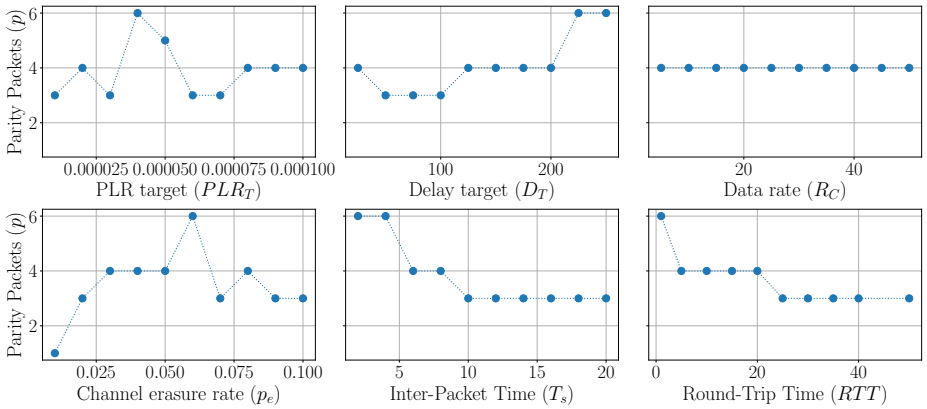
PRRT's ability to optimally adapt its performance to network changes is determined by how fast it is able to solve the optimization problem presented in Sec. 3.5. To the best of our knowledge, there is no known closed-form expression to obtain the optimal solution to the problem. The objective function and constraints defined in Eq. (3.10) are neither linear—except for the delay constraint—nor convex—see the binomial probabilities in Eq. (3.17) and Eq. (3.11). Therefore, mixed-integer linear/convex optimization cannot be applied to the optimization problem. Search algorithms have been proposed to explore the solution space until the optimal solution is found [11, 86]. However, the quantization of the variables k, p, and N_C produces a non-linear solution space that makes the efficient search for the optimal solution a challenging problem to solve.

Fig. 4.1 shows the optimal k, p, and N_C as functions of different channel (p_e, R_C, and RTT) and application parameters (D_T, PLR_T, and T_s). In every subplot, a different input parameter is linearly increased, while the remaining baseline conditions are kept constant ($PLR_T = 0.0001$, $D_T = 200$ ms, $R_C = 10$ Mbps, $p_e = 0.05$, $T_s = 4$ ms, and $RTT = 5$ ms). Although some output parameters have a relatively *smooth* behavior —i.e., they are either constant or monotonically increasing/decreasing—, small, linear increases in the input may produce sharp changes in the output in other cases. Bear in mind that, although the behaviors depicted cannot be directly generalized for any arbitrary input, Fig. 4.1 illustrates how irregular the output space can be. The precise description of the solution spaces requires a multifactorial analysis that is out of the scope of this thesis. Chapter 5 sheds some light on the impact of these non-linearities on the complexity of the problem at hand, whereas this chapter shows that exploring the entire solution space is computationally expensive and unfeasible even on high-end personal computers (see Sec. 4.2). Alternatively, greedy algorithms that give up optimality have been proposed in favor of a faster execution [86].

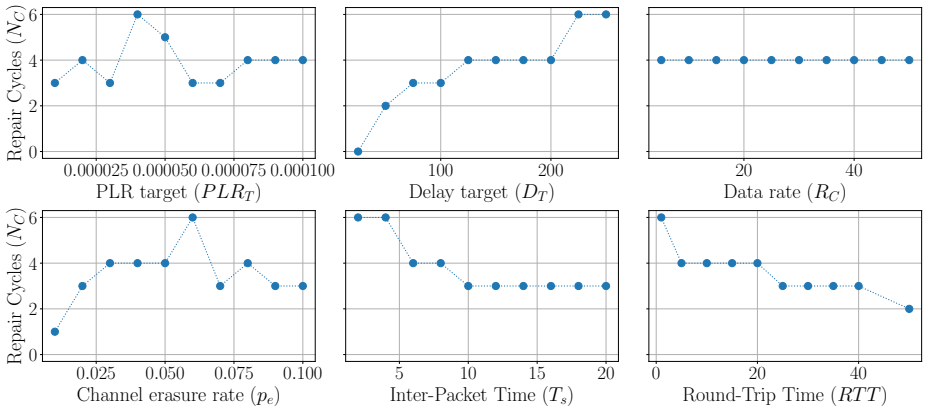
The PRRT protocol continuously senses the channel and executes the search algorithm whenever the conditions change in order to find the new optimal configuration or inform the application when no configuration exists that meets the constraints. There-



(a) Optimal block length



(b) Optimal number of parity packets



(c) Optimal number of repair cycles

Figure 4.1: Optimal k, p, and N_C as a function of channel and application parameters. In every subplot, a different input parameter is linearly increased, while the remaining baseline conditions are kept constant ($PLR_T = 0.0001$, $D_T = 200$ ms, $R_C = 10$ Mbps, $p_e = 0.05$, $T_s = 4$ ms, and $RTT = 5$ ms).

fore, the search algorithm has a double purpose: i) finding the optimal configuration according to a certain metric, and ii) ensuring that any configured configuration meets the application constraints. It can be shown that evaluating the constraints for a single configuration can be done relatively faster when compared to the complete search. However, if the channel change is sufficiently large so the protocol configuration no longer meets the constraints, the complete search algorithm must be executed. It is in this case that PRRT's performance guarantees are lost as the protocol does not know whether a configuration exists that meets the constraints until the search is complete. Ideally, the search execution time would be predictably low in order to minimize and upper bound the time the performance guarantees are lost. This predictable behavior is key for safety-critical CPSs, which demand a predictable channel to ensure their operation. The algorithm's inference time depends on how it treats the recursive dependencies among k, p, and N_C, which in turn depend on the application constraints—e.g., the larger the time budget, the more repair cycles could be configured, the number of required parity packets will depend on how wide the gap between the channel erasure rate and the target loss rate is, etc. This chapter introduces different search algorithms [11, 86] and compares their efficiency in the exploration of the solution space. The chapter also shows that the predictability of the search algorithm can be improved via the decomposition of the optimization into smaller sub-problems, which enables a more structured search that reduces the number of evaluated configurations by several orders of magnitude.

4.1 Predictable Communication Channel

Whenever the channel state changes, PRRT must solve the optimization problem in Sec. 3.5 to find a configuration that meets the application constraints. If no configuration is found, it informs the application that meeting its demands is not possible with the current channel. While the search algorithm is executing, the performance guarantees are lost as a result of the uncertainty regarding the algorithm's output.

Definition 4.1.1 (Channel Coherence Time)**.** The *Channel Coherence Time* (T_c) is the time period in which the channel can be assumed to be constant in the time domain, such that the amplitude of two samples of the channel are correlated [76].

Different formal definitions of channel coherence time exist in the literature [76], but a frequently used threshold is 0.5 of the channel autocorrelation function. The channel coherence can be expressed as a function of the maximum Doppler frequency (f_D) [76]: $T_c = \frac{1}{2f_D}$. Alternative functions exist in the literature for special Doppler spectra as well. Nevertheless, it is not the objective of this thesis to provide an in-depth analysis of the channel coherence time. Instead, we assume that $T_c \geq D_T$. Otherwise, no solution exists for the optimization problem in Eq. (3.10) that holds within the delay budget specified by the application.

Definition 4.1.2 (Protocol Reaction Time)**.** The *Protocol Reaction Time* (T_r) is the elapsed time between the protocol detecting a change in the channel state and it obtaining the optimal configuration from the search algorithm in Eq. (3.10).

For PRRT to provide a predictable communication channel to the application, the *hard* requirement $T_r \leq T_c$ must hold. Otherwise, the channel changes before a new configuration is found, making the protocol configuration oscillate without providing any guarantees. Adaptive HARQ configuration changes take place on a per-block basis—i.e., once the first packet in the block is transmitted, the configuration cannot be changed until a new block begins approximately D_T later. In other words, D_T establishes the lowest granularity at which the protocol may change its configuration. In the worst case, a channel state change is detected immediately before one of these configuration change opportunities, leaving no reaction time to execute the search before the end of the block. In order to limit the number of blocks that may not fulfill the constraints[1] to only one, the *soft* requirement $T_r < D_T$ should hold.

As PRRT is designed to support a wide range of application target delays (see Sec. 3.1), a stricter version of the aforementioned second requirement exists: the protocol reaction time should be below the target delay of the most stringent application the protocol is designed to support ($T_r < {D_T}^{min}$). This third requirement does not improve the predictability of the performance guarantees *per se*, which is already fulfilled with the previous two requirements, but it improves the protocol's non-functional properties as follows:

1. It reduces the resource footprint of the system due to the lower processing overhead, thereby enabling PRRT on the resource-constrained embedded platforms many CPSs are executed on [128, 159, 4].

2. It provides a more predictable execution time that CPSs can use for precise task scheduling [180]—e.g., in slowly changing channels with $T_c \gg T_r$, the search algorithm could be more sporadically executed in order to save resources, or in real-time systems where the unpredictable execution time of the search algorithm would negatively impact the performance of other tasks it shares the resources with.

The three aforementioned requirements for the protocol reaction time can be summarized as follows:

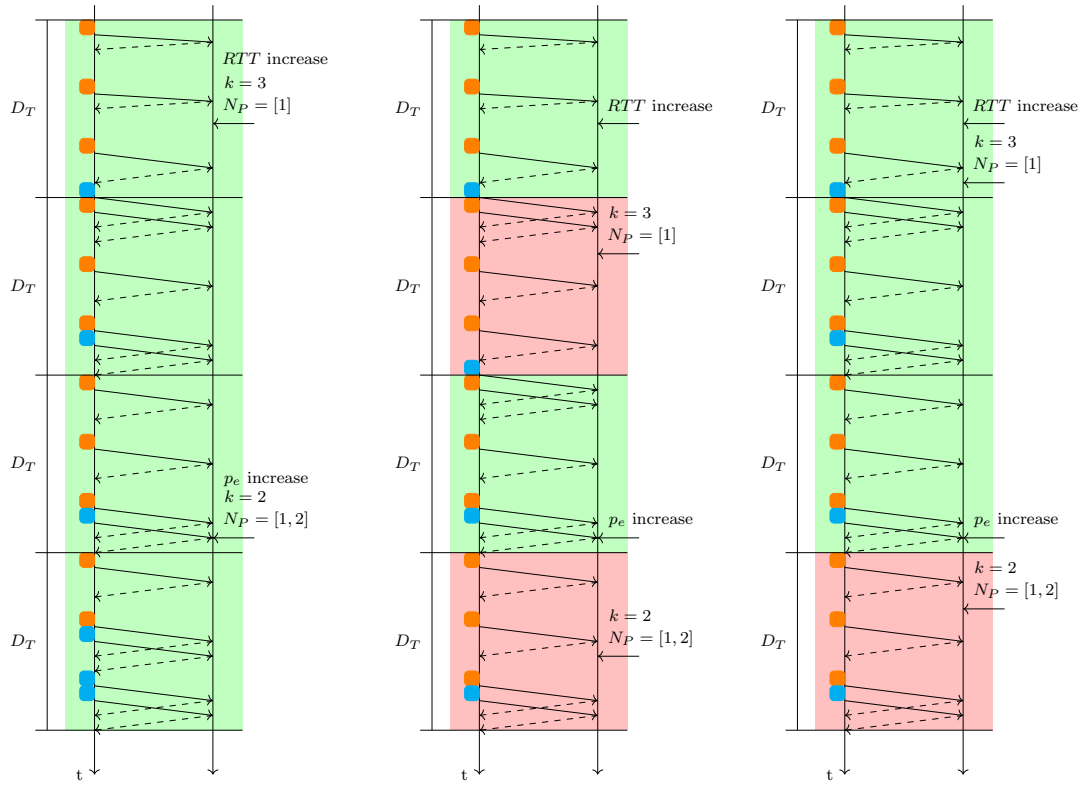
Requirement 4.1.1 ($T_r < T_c$)**.** The search algorithm MUST have a faster inference time than the channel coherence time.

Requirement 4.1.2 ($T_r < D_T$)**.** The search algorithm SHOULD have a faster inference time than D_T.

Requirement 4.1.3 ($T_r < {D_T}^{min}$)**.** The search algorithm MAY have a faster inference time than ${D_T}^{min}$.

The verbs MUST, SHOULD, and MAY here indicate requirement levels as defined by the Internet Engineering Task Force (IETF) [25]. Fig. 4.2 shows an example of

[1] Bear in mind that it may still be that the old configuration still meets the constraints for the new channel conditions, but this does not occur in a predictable manner.



(a) $T_r = 0$ **(b)** $0 < T_r < D_T$ **(c)** $0 < T_r < {D_T}^{min}$

Figure 4.2: Impact of the protocol reaction time on the predictability of the communication channel. The delay budget shades represent valid configurations (green), and invalid configurations (red).

the impact of Req. 4.1.2 and Req. 4.1.3 on the protocol's predictability. If none of the aforementioned requirements hold, the search algorithm is *non-performant*. If only Req. 4.1.1 holds for every input, the search algorithm is said to provide *soft* performance guarantees. The term "soft" here relates to the ability of the algorithm to eventually reach a valid configuration (see Def. 3.6.1) without an upper bound to how many blocks are impacted until it does so. If both Req. 4.1.1 and Req. 4.1.2 hold, the search algorithm is said to provide *strong* performance guarantees. The term "strong" here relates to the ability of the algorithm to reach a valid configuration (see the definition in Def. 3.6.1) before the end of the following block.

Fig. 4.2a represents the optimal case in which the search algorithm can be executed infinitely fast ($T_r = 0$), and the new optimal configuration is obtained as soon as a channel change is detected, which is applied by the protocol for the block immediately after the current one. The presented example considers two changes in the channel state. Firstly, the RTT is doubled in the middle of the first block so that the protocol switches the parity packet from the reactive to the proactive cycle in the later blocks. This

paradigmatic case clearly represents the concept of a valid configuration (see Def. 3.6.1). It can be observed that, due to the RTT increase, the parity packet arrives after the target delay and hence it cannot be used to correct any losses. Nevertheless, the configuration is still considered valid because it met the constraints for the channel state at the start of the block transmission. The second channel change is an increase in the channel erasure rate at the end of the third block. The protocol reacts to the new p_e by incrementing the number of parity packets from 1 to 3. To accommodate the extra redundancy, the block length is decreased by one, which in turn leaves enough time budget to use a reactive cycle.

If $T_r < D_T$, the protocol fails to fulfill the constraints in two blocks (see Fig. 4.2b). How often the protocol does not fulfill the constraints depends on how much time is left until the beginning of the next block since channel changes are detected. In the best case, the channel change is always at the beginning of a block and hence all blocks fulfill the constraints because $0 < T_r < D_T$. In the worst case, the channel change is always detected at the end of a block so that the number of invalid blocks equals the number of channel changes. Assume a channel change is detected $\epsilon \cdot D_T$ before the end of the target delay, a search algorithm that can react before the beginning of the next block is called a *ϵ-fulfilling algorithm* as defined in Def. 4.1.3.

Definition 4.1.3 (ϵ-fulfilling Algorithm)**.** A search algorithm is *ϵ-fulfilling* if, for any channel model $\mathcal{M}(p_e, RTT, R_C)$, its reaction time is $T_r \leq \epsilon \cdot D_T$ with $0 \leq \epsilon \leq 1$.

Given the application types PRRT is designed to support (see Sec. 3.1), for the third requirement to hold, the search inference time must be below the millisecond range for any channel and application. Although a faster reaction improves the performance predictability, there may still be cases in which the channel change is detected so close to the deadline that the following block cannot be adapted in time, as shown in the fourth block in Fig. 4.2c.

In the following, different search algorithms are presented and evaluated according to this definition of communication channel predictability.

4.2 Full Search

The *Full Search* proposed in [86] checks all possible combinations and outputs the one with the minimum RI. As PRRT operates in the Galois Field $GF(2^8)$, then $k, p, N_C \in [0, 255]$. While k and p are directly limited by the field size, p bounds the number of repair cycles—i.e., each cycle must have at least one parity packet—, and hence the three parameters are defined within the same range. A straightforward implementation of the Full Search would check all possible combinations $\forall\ k, p, N_C \in [0, 255]$. Nevertheless, the search space can be reduced by ignoring parameters that do not fulfill the delay constraint.

The number of repair cycles is upper bounded by $N_{C,max}$ (see Eq. (4.1)) the maximum number of repair cycles when pure ARQ is used with a single parity packet per round—i.e., $N_P[i] = 1 \ \ \forall\ i \in [1, N_C]$. Therefore, the HARQ delay must fulfill the in-

Algorithm 1 Full Search

Require: $N_{C,max}$
Ensure: $k^*, {N_P}^* = \arg\min_{k,N_P} RI(k, N_C, N_P)$

```
1:  k* ← 0
2:  N_P* ← 0
3:  ri* ← ∞
4:  for N_C = 0 → N_C,max do
5:      for k = 1 → k_max(N_C) do
6:          p ← derive_p(k)
7:          for N_P ∈ gen_compositions(p, N_C, 1, p − N_C + 1) do
8:              ri ← RI(k, N_C, N_P)
9:              if meets_constraints(k, N_P) ∧ ri < ri* then
10:                 k* ← k
11:                 N_P* ← N_P
12:                 ri* ← ri
13:             end if
14:         end for
15:     end for
16: end for
17: return (k*, N_P*)
```

equality $D_{HARQ}(1, [0, ones(N_C)]) \leq D_T$ to meet the delay constraint. Eq. (4.1) shows how the upper bound $N_{C,max}$ can be obtained. Once the number of repair cycles is known, the upper bound to the block length depends on how many parity packets can be collected within the remaining time budget that is not required to transmit the N_C cycles. The number of parity packets per cycle is again assumed to be one so that $D_{HARQ}(k_{max}, [0, ones(N_C)]) \leq D_T$ must hold true. Eq. (4.2) shows how the maximum block length can be obtained from the inequality above.

$$N_{C,max} = \left\lfloor \frac{D_T - D_{FEC}(1, [0, 1])}{D_{ARQ}([0, 1]) - \frac{RTT}{2}} \right\rfloor \tag{4.1}$$

$$k_{max}(N_C) = \left\lfloor \frac{D_T - \frac{RTT + D_{RS}}{2} - N_C \cdot D_{ARQ}([0, 1])}{T_s} \right\rfloor \tag{4.2}$$

The Full Search algorithm is shown in Alg. 1. For every (k, N_C) pair, the algorithm obtains p and all possible repair schedules N_P. The $derive\_p(k)$ in line 6 returns the optimal number of parity packets—i.e., the smallest p for k that meets the PLR constraint—, which is found increasing p from 0 to 255 until the optimal value is found.[2] How to optimally distribute those p packets among the N_C repair cycles corresponds

[2]The maximum codeword ($\mathbf{c}_{max}$) in a Vandermonde code in $GF(2^8)$ is 255 if the code is non-systematic and 510 if it is in the systematic form. In the latter case, the matrix is constructed by

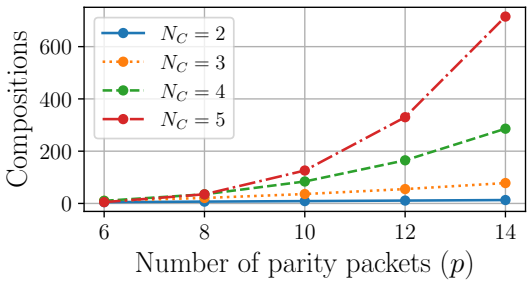


Figure 4.3: Number of restricted integer compositions by number of parity packets (p) and retransmission cycles (N_C).

to the well-known restricted integer compositions problem [69]: the integer p must be split into N_C components where the minimum possible value for any component is 1—at least one parity packet must be transmitted per cycle—and the maximum $p - N_C + 1$ so that no cycle is assigned too many packets. The *gen_compositions*(p, N_C, a, b) algorithm presented in [69] finds all the compositions of p into N_C sets with minimum and maximum values a and b, respectively, with linear complexity $\mathcal{O}(N_C)$. Therefore, the number of different configurations checked by Alg. 1 is

$$\mathcal{N}_{Full} = \sum_{N_C=0}^{N_{C,max}} \sum_{k=1}^{k_{max}} \sum_{i=0}^{p_{opt}-N_C} C\big(p_{opt} - i, N_C, 1, p_{opt} - i - N_C + 1\big) \tag{4.3}$$

where $C(p, N_C, a, b)$ is the number of restricted integer compositions and $p_{opt} = ||N_P{}^*||_1$ the optimal number of parity packets. Bear in mind that the FEC cycle may have, at most, $p_{opt} - N_C$ to ensure that all repair cycles are occupied, which is modeled by the most inner summation term. Fig. 4.3 depicts how $C(p, N_C, a, b)$ increases as a function of p and N_C. Already for a relatively small number of cycles, the number of partitions can reach the order of hundreds, suggesting that running the Full Search may be impractical even on commodity hardware, which is confirmed in Sec. 4.4.

4.3 SHARQ: Scheduled HARQ

The *Scheduled Hybrid Automatic Repeat reQuest* (SHARQ) [11] algorithm reduces the search inference time with the combination of i) a more efficient exploration of the three-dimensional solution space formed by k, p, and N_C, and ii) a Graph Search algorithm for a faster repair schedule construction.

appending the 255×255 identity matrix to the 255×255 Vandermonde matrix. Unless stated otherwise, this thesis assumes $\mathbf{c}_{max} = 255$ in order to reduce the search space and the coding complexity.

Algorithm 2 Derive number of parity packets

Require: k
Ensure: $p = derive\_p(k)$
1: $p_{min} = 0$
2: $p_{max} = 255 - k$
3: **while** $p_{max} - p_{min} > 1$ **do**
4: $\quad p_{mid} = p_{min} + round(\frac{p_{max} - p_{min}}{2})$
5: $\quad$ **if** $PLR_{HARQ}(k, [p_{mid}]) \leq PLR_T$ **then**
6: $\quad\quad p_{max} = p_{mid}$
7: $\quad$ **else**
8: $\quad\quad p_{min} = p_{mid}$
9: $\quad$ **end if**
10: **end while**
11: **return** p_{max}

4.3.1 Efficient Solution Space Exploration

SHARQ decouples the delay and loss rate constraints to reduce the computational complexity of the search. According to Eq. (3.17), the PLR constraint solely depends on k and p. In an i.i.d. channel, the probability of recovering the data packets does not depend on *when* the parity packets are sent, but on *how many* of them are transmitted. Although the optimal number of parity packets only depends on k (see Eq. (4.4)), Full Search evaluates Eq. (4.4) for every (k, N_C) pair that fulfills the delay constraint.

$$p_{opt}(k) = min\{p \,|\, PLR_{HARQ}(k, p) \leq PLR_T\} \tag{4.4}$$

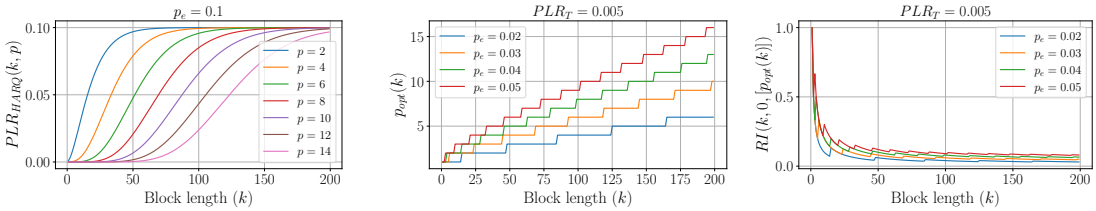
The set $K_{N_C} = \{k \in \mathbb{N} \mid 1 \leq k \leq k_{max}(N_C)\}$ corresponds to all k's that, given a number of repair cycles N_C, meet the delay constraint. As $k_{max}(N_C) \leq k_{max}(N_C + 1)$, then $K_{N_C} \subseteq K_{N_C+1}$ holds. Therefore, Full Search incurs a computational overhead due to the repeated evaluation of Eq. (4.4) for the same set of block lengths. This overhead is a direct consequence of the coupling between the delay and PLR constraints when the algorithm performs the joint search for k, p, and N_C.

$$k_{lim} = min\big(k_{lim}^{D_T}, k_{lim}^{PLR_T}\big) \tag{4.5}$$

$$k_{lim}^{D_T} = \left\lfloor \frac{D_T - \frac{RTT + D_{RS}}{2}}{T_s} \right\rfloor \tag{4.6}$$

$$k_{lim}^{PLR_T} = max\{k \,|\, PLR_{HARQ}(k, 255 - k) \leq PLR_T\} \tag{4.7}$$

Given the largest block length that fulfills the delay constraint (Eq. (4.6)) and the largest block length that meets the PLR constraint (Eq. (4.7)), the minimum of the two (k_{lim}, see Eq. (4.5)) is the largest block length that meets both constraints at the same time. The required number of parity packets to meet the PLR constraint with



(a) **(b)** **(c)**

Figure 4.4: For a fixed number of parity packets p, the packet loss rate $PLR_{HARQ}(k, p)$ and the optimal number of parity packets $p_{opt}(k)$ a monotonically increasing function of the block length k, whereas the redundancy information $(RI(k, N_C, N_P))$ is a non-convex function due to quantization effects as $k, p \in \mathbb{N}$.

k_{lim} can be obtained in logarithmic complexity with the binary search in Alg. 2. The binary search is only possible because $p_{opt}(k)$ is a monotonically increasing function: as k increases and p is kept constant, every parity packet carries information from more data packets. Therefore, every packet loss is more harmful as no extra parity packet is added, i.e., $PLR_{HARQ}(k, p) < PLR_{HARQ}(k+1, p)$. It directly follows that $p_{opt}(k) \leq p_{opt}(k+1)$, with the equality holding true when $PLR_{HARQ}(k+1, p_{opt}(k)) \leq PLR_T$. Fig. 4.4a depicts the monotonically increasing behavior of $PLR_{HARQ}(k, p)$. As expected, Fig. 4.4b shows that the larger the block length, the more parity packets are required to reach a PLR below a constant PLR_T. However, the fact that $p_{opt}(k) < p_{opt}(k+1)$ should not support the intuition that smaller block lengths are generally preferred due to their smaller RI. Although $RI(k, N_C, N_P)$ is a non-convex function due to quantization effects, there is a clear tendency towards smaller RIs the larger the block length is, as shown in Fig. 4.4c. Finally, Fig. 4.4b shows that several block lengths share the same p_{opt} and when it differs, the p_{opt} for neighboring k's is often increased just by one. Therefore, $derive\_p(k)$ need only be executed once for k_{lim}, and p_{opt} for smaller block lengths can be obtained with an exploration around the optimal p from the previous block length.

Once p is fixed by the PLR constraint, the delay constraint fully determines N_C. Fig. 4.5 shows that the probability of later cycles being triggered decreases with every new parity packet transmission. Every transmitted parity packet has a probability $1 - p_e$ of arriving at the receiver. When it is the k'th received packet for a block, PRRT's feedback mechanism prevents later cycles from being triggered (see Sec. 3.3). In other words, the transmission of at least one parity packet in every repair cycle decreases the probability of triggering later cycles. Therefore, the protocol should always use all the repair cycles that fit into the remaining time budget (see Eq. (4.8)).

$$N_{C,opt}(k, p) = min\left(p, \left\lfloor \frac{D_T - k \cdot T_s - p \cdot \frac{L_p}{R_C} - \frac{RTT + D_{RS}}{2}}{RTT + D_{RS} + D_{PL}} \right\rfloor\right) \tag{4.8}$$

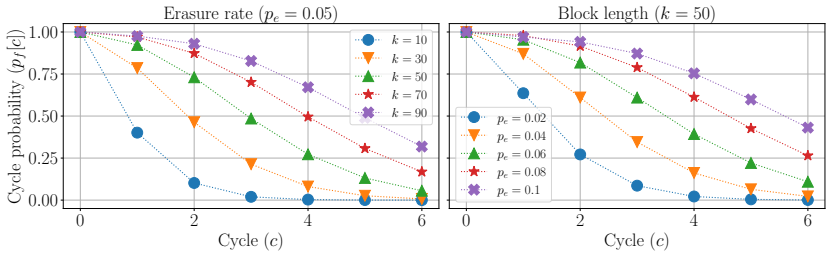


Figure 4.5: The probability of decreasing new repair cycles decreases with every parity packet transmission. The results here presented have been obtained with a repair schedule $N_P = [0, 1, 1, 1, 1, 1, 1]$.

$$\mathcal{N}_{Explore} = \sum_{k=1}^{k_{lim}} \sum_{i=0}^{p_{opt}-N_{C,opt}} C\big(p_{opt} - i, N_{C,opt}, 1, p_{opt} - i - N_{C,opt} + 1\big) \tag{4.9}$$

The *Fast Exploration* algorithm in Alg. 3 shows the integration of this more efficient solution space exploration into the Full Search algorithm. The algorithm has purely been included to independently analyze the two algorithmic optimizations introduced in SHARQ on the search efficiency. The number of configurations evaluated by Fast Exploration is given in Eq. (4.9), which still depends on the restricted integer compositions $C(p, N_C, a, b)$.

4.3.2 Fast Parity Packet Scheduling

This section introduces a *Graph Search*[3] that finds the optimal repair schedule in polynomial complexity, thereby reducing the number of evaluated configurations compared to the Full Search.

The probability of triggering a cycle $p_f[c]$—see Eq. (3.13)—solely depends on the *number* of parity packets in previous cycles ($\mathbf{n}[c-1]$), but not on *how* these packets are scheduled. Based on this observation, all possible schedules for a (p, N_C) tuple can be modeled with the graph in Eq. (4.10), where each path from the *start* node to (p, N_C) corresponds to a schedule. The edges are chosen to enforce that every retransmission cycle has at least one parity packet assigned to it—i.e., $N_P[c] > 0 \;\; \forall\, c \in [1, N_C]$. Fig. 4.6 depicts an example of the resulting graph for $(p, N_C) = (6, 3)$.

[3]This algorithm has been developed solely by Kai Vogelgesang and published in a joint conference article in 2023 [11]. I would like to thank Kai not only for this contribution but for all the discussions and exchanges of ideas over the last couple of years as well.

Algorithm 3 Fast Exploration

Require: k_{lim}
Ensure: $k^*, {N_P}^* = \arg\min_{k,N_P} RI(k, N_C, N_P)$

```
p ← p_opt(k_lim)
for k = k_lim → 1 do
    while PLR_HARQ(k, p − 1) ≤ PLR_T do
        p ← p − 1
    end while
    N_C ← N_C,opt(k, p)
    if N_C < 0 then
        continue
    end if
    for N_P ∈ gen_compositions(p, N_C, 1, p − N_C + 1) do
        ri ← RI(k, N_C, N_P)
        if meets_constraints(k, N_P) ∧ ri < ri* then
            k* ← k
            N_P* ← N_P
            ri* ← ri
        end if
    end for
end for
return (k*, N_P*)
```

$$
\begin{aligned}
G &= (V, E, w_E) \\
V &= \{start\} \cup \big\{(x, y) \mid 0 \le x \le p, 0 \le y \le N_C\big\} \\
E &= \big\{(start, (x, 1)) \mid 0 \le x \le p - N_C\big\} \\
&\quad \cup \big\{((x', y), (x, y+1)) \mid 0 \le y < N_C - 1 \\
&\qquad \wedge\, x - x' \ge 1 \wedge y \le x' \le p - N_C + y\big\} \\
&\quad \cup \big\{((x, N_C - 1), (p, N_C)) \mid N_C - 1 \le x < p\big\}
\end{aligned}
\tag{4.10}
$$

The edge weights, defined in Eq. (4.11), reflect the RI cost of assigning $x - x'$ parity packets to the y cycle. Given this weight definition, finding the schedule with minimum RI corresponds to finding the shortest path.

$$
\begin{aligned}
w_E(((x', y-1), (x, y))) &= \frac{1}{k} \cdot p_f^M[x'] \cdot (x - x'), \\
w_E((start, (x, 1))) &= \frac{1}{k} \cdot x
\end{aligned}
\tag{4.11}
$$

The graph can be computed following the dynamic programming approach presented in Alg. 4. To ensure that every repair cycle has at least one parity packet, no more than

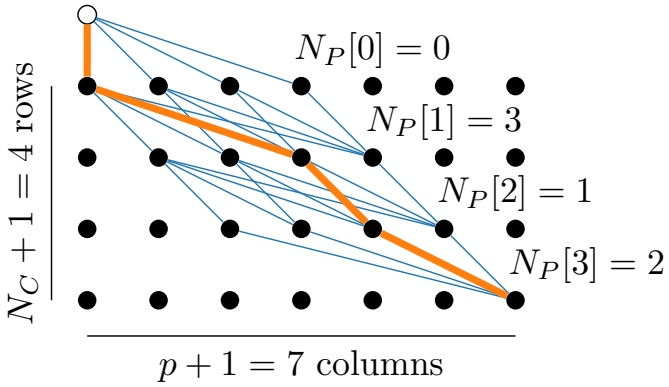


Figure 4.6: Graph for $p = 6$ and $N_C = 3$. Each path represents a choice of N_P. The edge weights are set such that the N_P with the lowest RI corresponds to the shortest path. The highlighted edges represent $N_P = [0, 3, 1, 2]$.

$p - N_C$ parity packets may be assigned in the 0th round and no repair cycle may have more than $p - N_C + 1$. The algorithm controls it with the *lower* and *upper* bound variables. The minimum distance for every vertex is stored in the $D[x, y]$ array, whereas the $Parent[x, y]$ array allows for reconstructing the optimal schedule at the end. Since Eq. (3.12) must be evaluated for every edge, the algorithm pre-computes the $\boldsymbol{pp}$ table, in which every entry is defined as $\boldsymbol{pp}[n, k] = \sum_{i=0}^{k} \binom{n}{i}(1 - p_e)^i {p_e}^{(n-i)}$. As a result, the probability $p_f[c]$ can be directly evaluated in $\mathcal{O}(1)$ with Eq. (4.12) and the edge weights in each layer can be obtained with $\mathcal{O}(p)$. Each layer has $\mathcal{O}(p)$ nodes with $\mathcal{O}(p)$ predecessors each. Since there are $\mathcal{O}(N_C)$ layers, the time complexity of the Graph Search algorithm is $\mathcal{O}(p^2 N_C)$.

$$p_f[c] = \boldsymbol{pp}[\mathbf{n}[c-1], k] - \boldsymbol{pp}[\mathbf{n}[c-1], max(0, \mathbf{n}[c-1] - p)] \tag{4.12}$$

The $\boldsymbol{pp}$ table must be recomputed when a new p_e is measured due to changes in the network conditions. Since it is a $2^m \times 2^m$ table, with $2^m = 256$ the size of the Galois Field employed in the MDS code (see Sec. 3.1.2), the table can be computed with polynomial complexity $\mathcal{O}(256^2)$ following the recursive expression in Eq. (4.13). This operation can be computed with polynomial complexity $\mathcal{O}(k^2)$ following the recursive expression in Eq. (4.13). The recursive loop is initialized with $\boldsymbol{dp}[0, 0] = 1$ and the $\boldsymbol{dp}[n, k]$ array is created for $n, k \leq 255$. Every column of the $\boldsymbol{pp}$ table is then iteratively calculated as $\boldsymbol{pp}[n, k] = \sum_{i=0}^{k} \boldsymbol{dp}[n, i]$.

$$\begin{aligned} \boldsymbol{dp}[n, k] &= \binom{n}{k} \cdot {p_e}^k \cdot (1 - p_e)^{n-k} \\ &= p_e \cdot \boldsymbol{dp}[n-1, i] + (1 - p_e) \cdot \boldsymbol{dp}[n-1, k-1] \end{aligned} \tag{4.13}$$

Algorithm 4 Graph Search

Require: k, p, N_C
Ensure: ${N_P}^* = \arg\min_{N_P} RI(k, N_C, N_P)$

$D[x,y] = \infty$; $Parent[x,y] = 0$ for $0 \leq x \leq p$ and $0 \leq y \leq N_C$
$lower \leftarrow 0$; $upper \leftarrow (p - N_C)$
for $x = lower \rightarrow upper$ **do**
    $D[x,0] \leftarrow x$; $Parent[x,0] \leftarrow x$
end for
for $y = 1 \rightarrow N_C$ **do**
    $lower++$; $upper++$
    for $x = lower \rightarrow upper$ **do**
        for $x' = (lower - 1) \rightarrow x - 1$ **do**
            $step \leftarrow x - x'$
            $current \leftarrow D[x', y-1] + step \cdot p_f^M[x']$
            if $current < D[x,y]$ **then**
                $D[x,y] \leftarrow current$
                $Parent[x,y] \leftarrow step$
            end if
        end for
    end for
end for
$x \leftarrow p$
for $y = N_C \rightarrow 0$ **do**
    $step \leftarrow Parent[x,y]$
    ${N_P}^*[y] \leftarrow step$
    $x \leftarrow x - step$
end for
$RI = \frac{1}{k} \cdot D[p, N_C]$
return $({N_P}^*, RI)$

$$\mathcal{N}_{Schedule} = \sum_{N_C=0}^{N_{C,max}} k_{max}(N_C) \tag{4.14}$$

The *Fast Schedule* algorithm presented in Alg. 5 follows a similar structure to the Full Search (see Sec. 4.2) but, instead of evaluating all possible schedules, obtains the optimal schedule from the Graph Search. As a result, a single schedule is considered for every k and N_C. Despite the significant reduction in the number of checked configurations in comparison to the Full Search (see Eq. (4.3) and Eq. (4.14)), the Fast Schedule algorithm still performs a suboptimal exploration of the solution space.

Algorithm 5 Fast Schedule

Require: $N_{C,max}$
Ensure: $k^*, {N_P}^* = \arg\min_{k,N_P} RI(k, N_C, N_P)$

```
1:  k* ← 0
2:  N_P* ← 0
3:  ri* ← ∞
4:  for N_C = 0 → N_C,max do
5:      for k = 1 → k_max(N_C) do
6:          p ← derive_p(k)
7:          (N_P, ri) ← graph_search(k, p, N_C)
8:          if meets_constraints(k, N_P) ∧ ri < ri* then
9:              k* ← k
10:             N_P* ← N_P
11:             ri* ← ri
12:         end if
13:     end for
14: end for
15: return (k*, N_P*)
```

4.3.3 The Algorithm

The SHARQ algorithm combines the benefits of a more efficient expiration (Fast Exploration) and a fast repair schedule construction (Fast Schedule) into a single algorithm (see Alg. 6).

$$\mathcal{N}_{SHARQ} = k_{lim} \tag{4.15}$$

SHARQ checks a single configuration in every iteration (See Eq. (4.15), which is the most efficient exploration among the presented algorithms. As shown in the following, this efficiency allows SHARQ to provide a predictable communication channel to the application when running on a desktop PC.

4.4 Search Comparison

This section compares the aforementioned search algorithms in terms of their efficiency in exploring the solution space. SHARQ significantly reduces the complexity of finding the optimal configuration in comparison to previous algorithms, thereby enabling a predictable transport for CPSs.

4.4.1 Methodology

Fundamentally, all the algorithms—i.e., Full Search, Fast Exploration, Fast Schedule, and SHARQ—provide the same solution to the optimization problem in Eq. (3.10).

Algorithm 6 SHARQ

Require: k_{lim}
Ensure: $k^*, {N_P}^* = \arg\min_{k,N_P} RI(k, N_C, N_P)$

```
p ← p_opt(k_lim)
for k = k_lim → 1 do
    while PLR_HARQ(k, p − 1) ≤ PLR_T do
        p ← p − 1
    end while
    N_C ← N_C,opt(k, p)
    if N_C < 0 then
        continue
    end if
    (N_P, RI) ← graph_search(k, p, N_C)
    if RI < RI(k*, N_C*, N_P*) then
        (k*, N_P*) ← (k, N_P)
    end if
end for
return (k*, N_P*)
```

Their main difference lies in how they explore the solution space until the optimal solution is found. The comparison of the four search algorithms has been based on three metrics: i) the *number of evaluated configurations* that are evaluated in an interaction of the algorithm,[4] ii) the *communication channel predictability*, which is based on the predictability requirements defined in Sec. 4.1, and iii) the *inference time* predictability over a dataset of realistic network traces. While the number of evaluated configurations provides a direct metric to evaluate the solution space exploration efficiency, the inference time of the search determines the protocol's predictability.

Unless otherwise stated, the inference times for the Fast Schedule and SHARQ algorithms include i) the *algorithmic search*, and ii) the ***pp*** *table computation*. The algorithmic search corresponds to Alg. 5, and Alg. 6 for the Fast Schedule, and SHARQ, respectively. The ***pp*** table computation depends on the parameter p_e and hence it must be re-computed when changes in the channel erasure rate are detected. Therefore, the ***pp*** table computation has been included in the measurements in order to obtain the worst-case execution time.

To analyze the performance of the algorithms under different network conditions, the dataset in Sec. 3.6 has been used for the inference time evaluation. A test dataset

[4]This metric should not be confused with the number of *valid* configurations reported in Sec. 3.6. The number of valid configurations refers to the ability of an error control scheme to provide configurations that meet the constraints. On the contrary, the number of *evaluated* configurations refers to the number of times a search algorithm must check whether a configuration meets the constraints or not. While the former measures the generalizability of an error control scheme, the latter measures the computational complexity of a search algorithm.

Table 4.1: Run-time complexity of the basic performance metrics that every search algorithm must evaluate to ensure the constraints are fulfilled and detect the optimal configuration.

Metric	Reference	Run-Time Complexity
Redundancy Information	Eq. (3.11)	$\mathcal{O}(k+p)$
Delay	Eq. (3.16)	$\mathcal{O}(1)$
Packet Loss Rate	Eq. (3.17)	$\mathcal{O}(k+log(p))$
Data Rate	Eq. (3.20)	$\mathcal{O}(k+p)$

consisting of 87,700 samples has been built in order to provide a faster evaluation of the algorithms. The test dataset avoids spurious cases whose solution space is empty by only containing inputs that are known to provide a valid configuration that fulfills all the constraints. Otherwise, the algorithm could directly output no configuration without performing any search, thereby resulting in an abnormally small inference time. Meaningful inference times are obtained by running the algorithms 50 times over the complete dataset in order to filter system noise such as interrupts to and from the CPU. The complete dataset is shifted by $\frac{1}{50}$ in order to avoid the samples always being executed in the same order in the dataset.

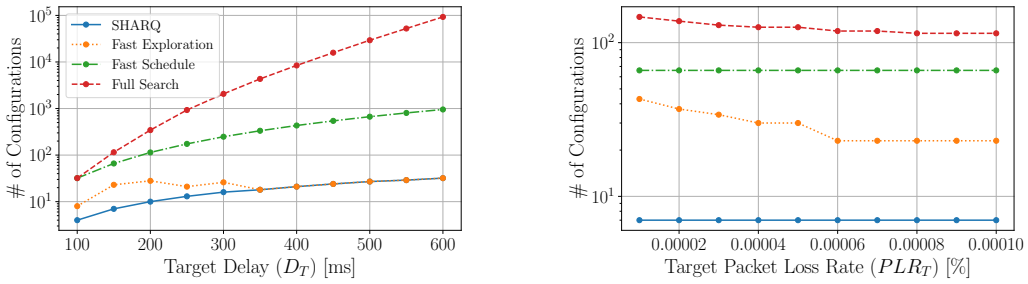
For every configuration, the algorithms evaluate the basic operations in Table 4.1 to check that the constraints are met and whether the configuration is the best one so far. Given how frequently these operations must be evaluated, the optimizations in Appendix A and Appendix B have been implemented to reduce their computational complexity. The search algorithms have been implemented in *Rust* and compiled with the `release` flag to enable compiler optimizations. The evaluations have been executed on two different devices. A *desktop PC* running Ubuntu 22.04.2 LTS with Linux kernel 5.19 on an Intel Core i7-7700 CPU at 3.6 GHz, and a *Raspberry Pi Zero W* running the Raspbian Buster operating system with Linux Kernel 4.19. These devices have been selected to cover both ends of the spectrum when it comes to computational power: while the former platform represents high-end devices such as edge servers where the controller in a distributed CPS may run [85], the latter is a resource-constrained platform that typically carries sensors and actuators [128, 159, 4].

4.4.2 Evaluation

In the following, solution space exploration efficiency and predictability results are presented for the different algorithms with the objective of analyzing their performance for CPS support.

Number of Evaluated Configurations

Fig. 4.7 compares the number of configurations the algorithms check for different target delays (Fig. 4.7a with $PLR_T = 0.0001$) and target PLR (Fig. 4.7b with $D_T = 150$ ms).



(a) Target Delay **(b)** Target Packet Loss Rate

Figure 4.7: Number of configurations evaluated by the search algorithms as a function of the target delay (D_T) and target packet loss rate (PLR_T). The baseline parameters are $R_C = 10$ Mbps, $p_e = 0.05$, $T_s = 4$ ms, $L_p = 1500$ B, $RTT = 20$ ms. In (a) $PLR_T = 0.0001$, and in (b) $D_T = 150$ ms are fixed.

The remaining parameters are the same for the two figures: $R_C = 10$ Mbps, $p_e = 0.05$, $T_s = 4$ ms, $L_p = 1500$ B, $RTT = 5$ ms.

For the depicted scenarios, the delay has a major impact on the size of the solution space than the PLR. A larger time budget increases the number of cycles to evaluate in Full Search and Fast Schedule (see Eq. (4.1)), whereas the largest block length may also increase in SHARQ and Fast Exploration (see Eq. (4.6))—i.e., in SHARQ it only increases if the PLR constraint allows, see Eq. (4.5) for more details. The inference time achieved by SHARQ is up to four orders of magnitude smaller than in Full Search in the high delay target regime. The major contributor to the fewer evaluated configurations is the structured search space exploration (see Sec. 4.3.1). While $p = N_C$—i.e., in the largest five delay targets—the Fast Exploration must only check one restricted integer composition per block length, and hence it explores the same solution space explored by SHARQ. If $p \neq N_C$, the restricted integer compositions increase but, for this particular case, Fast Exploration still checks fewer configurations than Fast Schedule. Despite the similar performance to Full Search in the low D_T regime, the impact of evaluating fewer schedules becomes visible in the high D_T regime, with Fast Schedule evaluating two orders of magnitude fewer configurations.

Changes in the PLR_T directly impact the number of parity packets p, which in turn may modify k and N_C in order to accommodate the extra parity packets. However, the latter is unlikely in current networks due to their high data rate (see Table 3.2). On the one hand, Fast Schedule's solution space is only affected if transmitting more parity packets can only be done at the expense of a retransmission cycle (see Eq. (4.14)). Its constant number of configurations in Fig. 4.7b suggest that N_C remains unchanged. On the other hand, the constant number of configurations for SHARQ means that the block length is also unchanged (see Eq. (4.5)). Therefore, the changes experienced by Full Search and Fast Exploration are a result of more packets to distribute among the repair cycles, as more parity packets are required the more stringent the PLR constraint is.

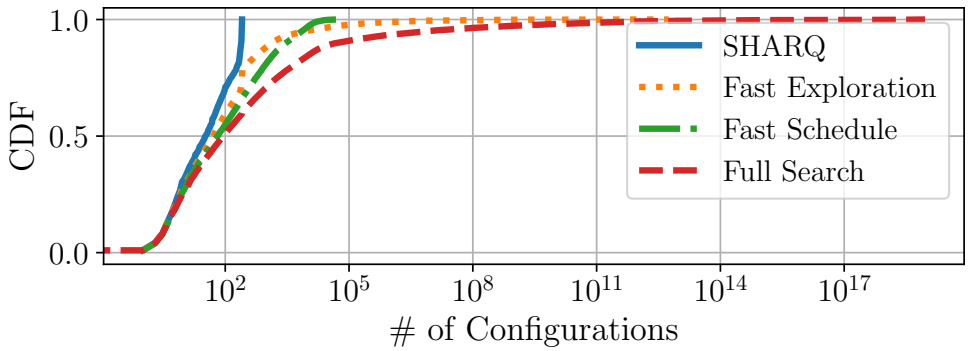


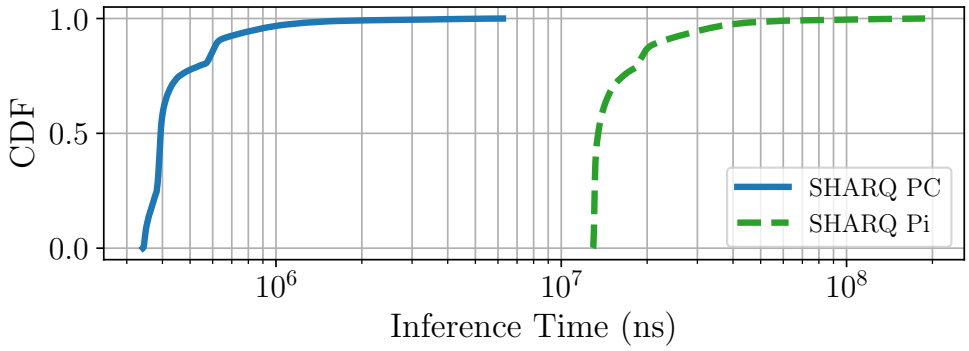
Figure 4.8: CDF of the number of configurations evaluated by every algorithm. The x-axis is on a logarithmic scale.

These are some examples of how intricate the relations among channel, application, and HARQ parameters are. To understand how the algorithms perform over a wider range of conditions, the number of configurations they evaluate has been obtained for every sample in the test dataset (see Fig. 4.8). All the algorithms evaluate a similar number of configurations for the first quartile of the dataset. SHARQ outperforms all the other algorithms since its upper limit to the number of configurations is $k_{lim} \leq 255$ (see Eq. (4.15)). Fast Exploration and Fast Schedule are relatively close except for the last quartile. While Fast Exploration evaluates fewer configurations than Fast Schedule up to the 94th percentile, its tail is 9 orders of magnitude larger. Although few samples in the dataset benefit from the Graph Search, fast scheduling plays a major role in the tail latency—i.e., inference time predictability—of the search algorithm. SHARQ has reduced the maximum evaluated configurations by 16 orders of magnitude without sacrificing RI optimality. Whether it is able to do it in real-time and with high predictability depends on the device-dependent inference time, which is evaluated in the following.

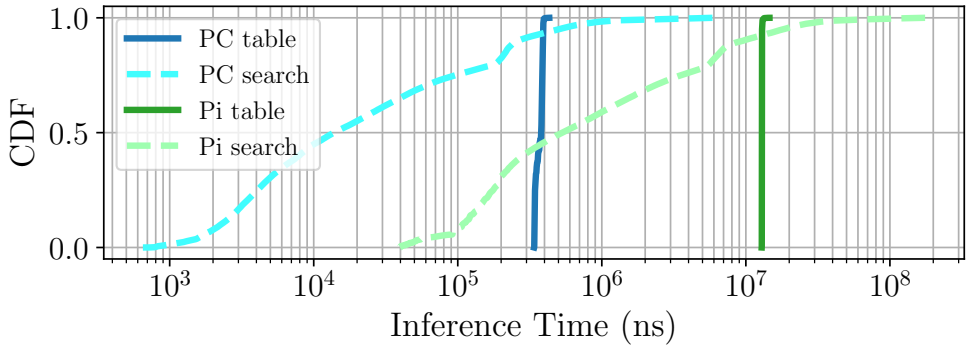
Inference Time

Fig. 4.9a depicts the average inference time over the 50 runs of the test dataset. When executed on the PC, SHARQ maintains the inference time below the millisecond mark up to the 96th percentile. However, the inference time is in the ten-millisecond range up to the 99th percentile on the Raspberry Pi. Despite the long tail, both curves show predictable behavior for the majority of the samples. On the PC, the inference time deviates less than 10% from the mean in 79.24% of the cases, whereas the percentage increases to 80.56% of the samples on the more predictable Pi.

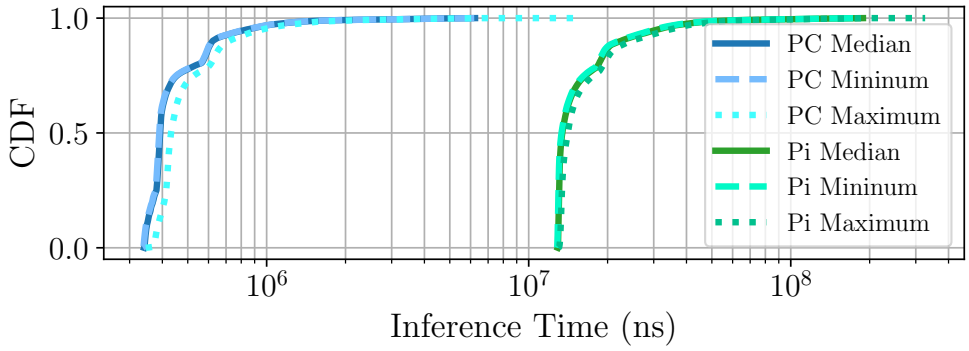
To shed some light on where the predictability comes from, Fig. 4.9b depicts the split between algorithmic search and ***pp*** table. The ***pp*** table construction marks the absolute lower bound to the inference time at 0.3 and 15 milliseconds on the PC and Pi, respectively. The table always has the same size and the p_e value plays no role



(a) CDF of the average inference time on the PC and the Raspberry Pi Zero W.



(b) CDF of the inference time split into ***pp*** table construction and algorithmic search.



(c) CDF of the mean, minimum, and maximum inference time over the 50 executions of the test dataset.

Figure 4.9: SHARQ inference time analysis.

in its computational complexity (see Sec. 4.3.2). Therefore, the ***pp*** table construction always entails the same number of operations. This predictability, in addition to the fact that it dominates the delay budget in 92.4% of the samples in both platforms, explains the observed predictable inference time in Fig. 4.9a. Not only is the inference time predictable across samples but across executions of the same sample as well. Fig. 4.9c depicts the mean, minimum, and maximum inference time out of the 50 runs over the test dataset. Although the search seems more sensitive to operative system noise in the sub-millisecond range when executed on the PC, the variance across executions is negligible outside of this band. As expected, a lower variance is perceived on the Raspberry Pi due to the more controlled execution environment as fewer services are running on its operating system that could interfere with the experiments.

It is important to note that predictability in this section solely refers to the expected inference time across different inputs to the algorithm and not to the ability of the algorithm to ensure the predictability of the communication channel. The latter predictability aspect will be analyzed in the following section.

Communication Channel Predictability

When it comes to the predictability of the communication channel, the search algorithm must fulfill the requirements in Sec. 4.1. Independently of the underlying channel coherence time, PRRT collects packets for a time window of 10 RTTs before applying any estimation filter. Therefore, the channel state is updated on an RTT basis: in the worst case, for fast-changing channels, PRRT's channel state is updated every round-trip time ($T_r \leq RTT$). The results presented in this section also consider a case in which no channel change is detected within the complete estimation window ($T_r \leq 10 \cdot RTT$).

Table 4.2 shows for how many samples in the test dataset the SHARQ algorithm meets the requirements. The algorithm always reacts to channel changes within the time budget—soft requirement in Sec. 4.1 or fourth column in Table 4.2. In fast-changing channels—i.e., second column—, SHARQ cannot provide predictable communication in 0.31% of the cases even when it runs on a desktop PC. On the resource-constrained Pi, the number of unpredictable channels increases to 63.38%. On the contrary, when the channel is stable—i.e., the third column—full predictability is achieved on the PC, while coverage of 91.77% is achieved on the Pi. Thanks to the fast execution of the PC, SHARQ is able to fulfill the most stringent of the requirements (Req. 4.1.3) up to the 99th percentile. However, it does not achieve the same low resource footprint and predictable inference time on the Pi.

The ability of the algorithms to channel changes within the current block is analyzed in Table 4.3. When executing on a desktop PC, SHARQ can react to channel changes without impacting later blocks even when the channel changes occur within a fourth of the target delay. On the other hand, on a more constrained platform, the predictability sharply decreases below 50% of the samples once 10% of the target delay is reached.

Device	Requirement			
	$T_r < RTT$	$T_r < 10 \cdot RTT$	$T_r < D_T$	$T_r < {D_T}^{min}$
PC	99.69%	100%	100%	99.17%
Pi	36.62%	91.77%	92.07%	0%

Table 4.2: SHARQ percentage of inferences on the test dataset meeting the predictability requirements.

Device	ϵ					
	0.75	**0.5**	**0.25**	**0.1**	**0.05**	**0.01**
PC	100%	100%	100%	99.68%	96.51%	84.12%
Pi	91.81%	88.87%	74.72%	44.6%	27.86%	0%

Table 4.3: SHARQ percentage of inferences on the test dataset meeting the ϵ-fulfilling requirement.

4.5 Discussion

This chapter has introduced two different predictability concepts that are relevant for error control in transport protocols in order to support CPSs. Firstly, the predictability of the communication channel offered to the application is the protocol's ability to timely adapt its configuration to cope with performance changes in the underlying physical channel so that the application constraints are always fulfilled. Sec. 4.1 has shown that this first predictability type depends on how the search inference time relates to the channel coherence time and the application target delay. Secondly, inference time predictability enables precise task scheduling in CPSs. As the size of the explored solution space depends on the application and channel parameters, achieving this second predictability type is a challenging task. This section discusses SHARQ's performance with regard to both predictability types.

Communication Channel Predictability

SHARQ combines the Graph Search presented in Sec. 4.3.2 for optimal repair schedule construction in polynomial time, with the structured search of the solution space in Sec. 4.3.1 to achieve a more predictable and faster reaction to channel changes. The evaluations presented in this chapter have shown that SHARQ is the first search algorithm that solves the optimization problem in Sec. 3.5 within the target delay of the application. In other words, SHARQ is the first algorithm providing a predictable optimal communication channel to the application. However, this performance is only achievable on a relatively powerful desktop PC. When the algorithm runs on a resource-constrained device, its predictability vanishes for a high percentage of the channels it is supposed to operate with. Although SHARQ is a first step toward supporting CPSs with PRRT, the tail inference time of the search algorithm must be reduced by several orders of

magnitude in order to extend the predictability to resource-constrained devices, which are the natural component of CPSs.

Despite its high predictability, SHARQ's long tail in the inference time prevents it from achieving high predictability on fast-changing channels. Being unable to react before the beginning of the following block has a different impact on the protocol performance depending on how stable the channel is. When the channel state remains constant for long periods, the impact of a slow reaction can be spread out over more blocks than if the channel state quickly changes again. SHARQ's poor performance on the resource-constrained Pi precludes its deployability on fast-changing channels. These channels are very common in high-mobility scenarios in which the sender, the receiver, or both may change their position in any dimension, which is a very common use case in CPSs ([4, 78, 110, 128, 159, 185, 226]). Another aspect to consider in fast-changing channels is the granularity of the configuration adaptation function. As the protocol performs adaptation on a per-block basis, the faster a block is transmitted the more opportunities there will be for changes. Therefore, not only should the search be quick enough to find the configuration before the next block, but the protocol may favor blocks that do not consume the complete time budget—e.g., limiting the block length, or the number of repair cycles.

For PRRT to support high predictability on resource-constrained devices and fast-changing channels, a common requisite for the search algorithm exists: *the inference time of the search algorithm must be reduced.* The inference time results presented in Sec. 4.4 point in two different directions. First, the large tail latency results from the algorithmic search, which only explores one dimension in SHARQ, namely the block length (see Eq. (4.15)). Therefore, further algorithmic optimizations are required to reduce the tail latency. Secondly, the ***pp*** table construction currently is the lower bound to the inference time. It is an open question whether optimal scheduling can be provided on embedded devices with limited computational power and, in case it is not, what the impact on the overall protocol performance is.

Predictably Low Inference Time

This chapter has also extended the predictability concept to the inference time of the search algorithm. This second concept represents a non-functional requirement of the protocol that does not impact its main purpose, namely providing data transport with predictable end-to-end latency and reliability. On the contrary, a predictable execution time allows for precise task scheduling, which is a broad topic on its own that spans real-time operating systems [56], multi-core CPU management [162], or precise timers [180], among others. This precise task execution concept has already been applied to transport layer function design in congestion control [161]. However, this chapter has focused on its applicability to error control design.

The observed predictability in the inference time is a direct consequence of the dynamic programming approach followed in SHARQ with the ***pp*** table construction. Nevertheless, SHARQ's predictability is not high enough yet for precise task scheduling due to its long tail latency. In addition, the section above proposes to omit the ***pp*** table to

increase the predictability of the communication channel. However, not constructing the ***pp*** table would increase the unpredictability of the inference time, as shown in Fig. 4.9b. Ideally, the search algorithm should have a predictable *low* inference time so that it can predictably react to channel changes in a wide range of devices, and thus the inference time predictability should not be dismissed.

4.6 Related Work

Error control in transport protocols has typically been approached with loss detection and the retransmission of loss packets (see Sec. 3.1). This reactive mechanism allows transport protocols to approach the channel capacity as long as the loss detection does not produce misclassifications. TCP's Fast Retransmit [27] detects packet losses upon the reception of three duplicated ACKs. However, this loss detection mechanism performs poorly with bursts of lost packets because duplicated ACKs only detect the first loss in the burst. Selective Acknowledgments (SACKs) [24] have been proposed to solve this issue by providing precise information about the received packets. The most recent proposal is RACK-TLP [187], which transmits less redundancy than previous approaches in the presence of packet reordering and reacts faster to packet losses in application-limited scenarios. On the one hand, Recent Acknowledgment (RACK) performs a time-based loss detection for a more robust detection than duplicated ACKs. Packet losses are detected upon a timer expiration. The timer accounts for the RTT elapsed between the packet transmission and its expected ACK arrival time, plus a configurable window that accounts for possible reordering that would delay the ACK. On the other hand, Tail Loss Probe (TLP) proactively retransmits packets in application-limited scenarios so that, when RACK does not operate because of the sparse ACKs, losses can be detected faster.

As more transport protocols implement FEC, the amount of transmitted redundancy depends on the algorithm that detects how many parity packets are transmitted instead of the loss detection mechanism. These algorithms must predict losses before they occur in order to decide how much redundancy must be put into the channel. Therefore, they observe the channel and build models to predict future channel dynamics. Originally, the proposed algorithms used relatively simple metrics to decide how much RI to transmit [41, 58, 138, 151]. Tickoo et al. [41] proposed a loss-tolerant TCP version that distinguishes congestion-caused losses—followed by packets marked by ECN—and losses caused by the poor quality of the communication channel. Such a distinction is instrumental in TCP to avoid the throughput drop—the default behavior when congestion is detected—when losses appear in wireless channels. The transmitted proactive redundancy is proportional to the estimated channel erasure rate, whereas reactive redundancy is still transmitted when the receiver does not get enough packets to decode the block. The authors show that this combination of proactive and reactive redundancy achieves higher throughput over lossy links than standard TCP. Ahmad et al. [58] implement a pure FEC scheme with rateless codes for time-constrained scenarios. The protocol implements a feedback mechanism that signals the correct decoding of each

block. The RI is proportional to the measured packet loss rate, which is adjusted when no ACK is received in order to increase the probability of the next block being decoded. Adaptive FEC has also been applied to time-sensitive applications to avoid head-of-line blocking when transmitting over Multipath TCP (MPTCP) [138]. The authors implement a zero-RTT, XOR-based FEC scheme that dynamically adjusts the transmitted redundancy to achieve a target residual loss rate. A similar dynamic approach has been implemented in QUIC as well [151].

The aforementioned solutions have been proposed for fully reliable transport, and hence they complement proactive redundancy, which achieves zero-RTT loss recovery, with reactive redundancy to ensure full reliability. However, similarly to PRRT, more recent approaches exploit the proactive and reactive combination to reduce the transmitted redundancy [154, 172, 188, 211]. This trend towards more efficient error control has motivated the development of more advanced metrics and channel models to fine-tune the transmitted RI. [154] adds a coding layer between IP and MPTCP to reduce the FCT of short flows in datacenter communication by avoiding head-of-line blocking. The proposed mechanism jointly balances the traffic overhead—i.e., the transmitted RI—and decoding delay—i.e., time to obtain enough packets to decode a block—by transmitting more parity packets in those flows with a higher head-of-line blocking probability. Cohen et al. [172] also aim at reducing the in-order delivery in fully reliable transport with an adaptive HARQ scheme based on Random Linear Network Coding (RLNC). A receiver feedback mechanism is used to estimate the erasure probability, channel variance, and expected loss burst pattern. These parameters are used to find the optimal balance between a prior and a posterior redundancy such that the transmitted RI is minimized. Based on the observed loss patterns in the channel, [188] adaptively parametrize a streaming code with two parameters: the maximum recoverable burst length, and the maximum number of arbitrary losses. To the best of our knowledge, the most similar search solution is proposed in [211]. The authors implement RLNC into QUIC and propose an adaptive search algorithm for optimal RI scheduling. The proposed scheme takes as an input the applications' sensitivity to delay, as well as how desirable it is for the application to rely on the proactive FEC cycle for a faster loss recovery. The resulting QUIC version is able to provide different reliability levels according to the application demands.

4.7 Conclusion

In order for PRRT's performance to be predictable—i.e., the D_T and PLR_T constraints configured by the application are fulfilled with a high probability—, it must timely react to channel changes. Designing an algorithm that solves Eq. (3.10) is challenging because traditional optimization tools cannot be applied. This chapter has shown that performing a Full Search of the solution space is unfeasible due to the large number of configurations to evaluate. The search algorithm must efficiently explore the solution spaced formed by k, p, and N_C, and quickly find the optimal schedule N_P. SHARQ is, to the best of our knowledge, the first search algorithm capable of solving Eq. (3.10)

in real-time. This algorithm involves a more structured search of the solution space that, with more efficient use of the application constraints, avoids evaluating the same solution several times. In addition, thanks to the Graph Search in Sec. 4.3.2, it can find the optimal schedule in polynomial time. The evaluations presented in this chapter have shown that SHARQ needs to evaluate significantly fewer configurations than the Full Search to find the optimum. As a result, SHARQ is able to provide a predictable communication channel to the application when executed on a computationally powerful desktop PC. However, it still fails to achieve similar predictability levels on resource-constrained devices. Therefore, in order to bring PRRT's performance guarantees to embedded devices with limited computational power, which are the natural component of CPSs, the computational complexity of the adaptive error control function must be further reduced.

Chapter 5

Deep Learning for Adaptive HARQ

The algorithmic optimizations presented in Chapter 4 have reduced the tail latency of finding the optimal HARQ configuration by several orders of magnitude. Despite its more predictable and lower inference time, SHARQ still fails to provide full predictability even on a computationally powerful desktop PC, not to mention its poor predictability on resource-constrained devices. This chapter departs from a purely algorithmic approach and explores learning-based solutions to the problem, which enable predictable performance on resource-constrained devices for the first time.

5.1 Learning How to Balance

Advances in algorithms, hardware, and software frameworks have led to more powerful neural networks, and hence Deep Learning (DL) has permeated almost every field as of today, including protocol design. The efficient feature extraction of Deep Neural Networks (DNNs) has enabled resource-constrained embedded devices to solve complex problems that were previously limited to more powerful computers. This section introduces the fundamentals of DL and how they could be leveraged to design more efficient search algorithms capable of providing predictable communication channels even on embedded devices.

5.1.1 Deep Learning Fundamentals

DNNs arguably are the most common machine learning models nowadays. Their goal is to approximate some mathematical function f^*. For example, given some classifier $y = f^*(\boldsymbol{x})$ that maps the input vector $\boldsymbol{x}$ to the class y, the neural network defines a mapping $\hat{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$. The parameters $\boldsymbol{\theta}$ are learned to provide the best function approximation according to a loss function $\mathcal{L}(y, \hat{y})$. Common loss functions are the Mean Absolute Error (MAE) or Mean Squared Error (MSE) for regression problems, or the cross-entropy function for classification problems.

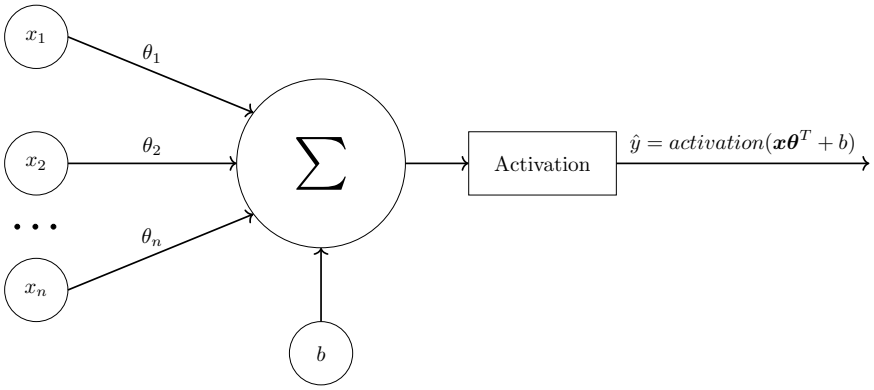


Figure 5.1: Artificial neuron architecture [207]

The most basic component of DNNs is the *artificial neuron*, whose architecture is depicted in Fig. 5.1. The artificial neuron applies a linear combination of the inputs, weighted by the parameters θ_i. If the summation of the inputs is above a certain threshold determined by the *bias* b and *activation function*, the artificial neuron triggers and propagates its output further. Nowadays, the most frequently used activation functions are the Rectified Linear Unit (ReLU) (see Eq. (5.1)) and sigmoid (see Eq. (5.2)) functions. Multiple neurons are collected into a *layer*, to which an input and output layer could be prepended and appended, respectively, in order to form a *neural network*.

$$ReLU(x) = max(0, x) \tag{5.1}$$

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{5.2}$$

An important concept for neural networks is their *depth*: when the network has more than one layer—a.k.a. *hidden layers*—, then it is known as a *deep neural network*. It can be mathematically proved that DNNs can approximate any mathematical function [20], which explains the broad generalizability of these models. As a result, DL has gained traction in nearly every field in recent years [221].

Being able to approximate any mathematical function is the same as saying that neural networks are non-linear functions, which is at the same time one of the major disadvantages of neural networks. Non-linearities cause the loss functions that are frequently used in other machine learning algorithms to be non-convex [114]. Therefore, the gradient descent algorithm has no convergence guarantees for neural networks. DNNs are trained with iterative, gradient-based optimizers that explore the solution space to avoid local minima far from the optimum [103]. For typical training, the dataset is split into batches, and every sample in the batch is fed into the neural network in what is called the forward pass. Once the complete batch has been processed, the loss is computed at the output of the network, and the back-propagation algorithm distributes this information in the backward pass by computing the gradient for each layer. The

parameters $\boldsymbol{\theta}$ are usually updated after every batch to reduce the memory footprint for large models. One iteration over the complete dataset is known as an epoch.

The hyperparameter tuning phase is instrumental in obtaining accurate models due to the lack of convergence guarantees. If hyperparameters—e.g., learning rate, number of layers and neurons per layer, batch size, number of epochs, weight initialization, etc.—are not correctly configured, the optimizer will not be able to bring the loss to a low value [164]. The wrong set of hyperparameters can also lead to overfitting: matching the training set so well that the neural network does not generalize for data it has not seen while training. A common practice to detect overfitting early in the training phase is splitting the dataset into the training, validation, and test dataset. Only the training dataset is used to update the weights. When the loss for the training dataset is significantly lower than for the validation dataset, the model is overfitting and the hyperparameters must be adjusted. The test dataset is a final check to avoid overfitting with data that has not been used in the training phase, neither for the weight tuning (training set) nor for the hyperparameter tuning (validation set). A common split is 60%, 20%, and 20% for the training, validation, and test dataset, respectively.

Advances in algorithms, hardware, and software frameworks have enabled the training of deeper models with more layers and parameters, and neural networks have become more accurate and capable, which has led to DNNs outperforming traditional methods in a plethora of use cases. On the one hand, large DNNs have motivated a performance leap in fields such as visual computing [90] or natural language processing [170]. On the other hand, larger models demand greater computational resources, with an increase in the energy demand and training costs that it entails, both at training and inference time [165, 181]. The training time for the best-performing models has experienced an exponential increase in the last decade [181], which has raised sustainability [165, 181, 227] and reproducibility [194] concerns from the research community.

5.1.2 Sustainable Deep Learning on Embedded Devices

The lack of a carbon-neutral energy grid makes power-hungry neural networks responsible for a significant portion of greenhouse emissions. Strubell et al. [165] estimate that training a single unit of these models can emit as much CO_2 as approximately 5 cars throughout their complete lifetime, including fuel. Another important aspect that is frequently overlooked is the competition for limited energy resources. Even if CO_2-neutral energy sources were available, the question remains whether those energy resources may be better allocated for other purposes instead. Improving the energy efficiency of deep learning is of the utmost importance in order to reduce its environmental impact. The GreenAI concept refers to the set of good practices leading to the development of a sustainable AI ecosystem in a broad sense [181]—i.e., developing sustainable practices in every stage of the pipeline. A wide range of proposals around this GreenAI paradigm have appeared in the last years. Energy-efficiency metrics should be reported for each model's training and hyperparameter tuning phases so that researchers take energy efficiency into account alongside accuracy when designing new models [181]. Some researchers have gone a step further, making the hyperparameter tuning carbon-

aware [227] to favor search architecture strategies based on their carbon intensity. Power draw models have been proposed to estimate the energy demand at inference time when executing the model on CPUs and GPUs [117], or TPUs [208]. At the core of this thesis is the sustainable design and deployment of CPSs, as discussed in Sec. 2.2.1. Therefore, if PRRT is extended with any DL component, it should be done following the GreenAI guidelines.

The remarkable success of DNNs has led to their introduction into embedded devices [135, 167, 192, 217], despite their limited computational power and memory size. The design of specific-purpose hardware, such as Google Coral Edge TPU[1] or NVIDIA Jetson Nano[2], has paved the way with faster and more efficient inference than is possible on edge CPUs, whereas TinyML has emerged to reduce neural network sizes to fit into low-power embedded devices without a significant accuracy drop. Most TinyML models are conceived as a compressed version of previously trained models. For example, *weight pruning* [118, 133] avoids executing weights that have little impact on the model's accuracy, *weight quantization* converts floating point weights into integers to operate with integer-only arithmetic and reduce the model size [115, 141], or *knowledge distillation* [42, 96] compresses a trained model into a smaller one without a significant loss of performance. While TinyML models are not necessarily "green"—e.g., if compressing the model requires more resources than what is saved at inference time—, both approaches have synergies that are instrumental for the sustainable deployment of data-driven systems at scale. Given SHARQ's poor performance on embedded devices, and the potential offered by deep learning, TinyML seems an ideal candidate for the design of a more efficient search algorithm.

5.1.3 Learning-based, Predictable Error Control

Machine learning algorithms can learn arbitrary functions if the architecture of the algorithm is powerful enough. In other words, the size of the DNN depends on the complexity of the problem to be learned. The term complexity here does not have the traditional meaning of "computational complexity" as expressed with the big-O notation, but it is related to the *learnability* of the problem.

Definition 5.1.1 (Learning complexity)**.** Given two functions $f_1^*(\boldsymbol{x})$ and $f_2^*(\boldsymbol{x})$, and their learned counterparts $f_1(\boldsymbol{x};\boldsymbol{\theta})$ and $f_2(\boldsymbol{x};\boldsymbol{\theta})$, respectively. The function $f_1^*(\boldsymbol{x})$ has a higher learning complexity than $f_2^*(\boldsymbol{x})$ if $f_1(\boldsymbol{x};\boldsymbol{\theta})$ requires more floating point operations than $f_2(\boldsymbol{x};\boldsymbol{\theta})$ to achieve the same accuracy level.

The *learning complexity* metric defined in Def. 5.1.1 should be carefully used in order to avoid unfair comparisons. For example, comparing a Convolutional Neural Network (CNN) and a network with fully connected layers both solving the same visual computing problem would be unfair because CNNs have been specifically designed to better

[1] `https://www.coral.ai/products/accelerator` (accessed January 23rd 2024)

[2] `https://developer.nvidia.com/embedded/jetson-nano-developer-kit` (accessed January 23rd 2024)

extract features for such kind of problems by exploring spatial relations between neighboring pixels. The learning complexity of the problem remains constant but the CNN likely requires fewer floating point operations than the network with fully connected layers. This example belongs to what is called Neural Architecture Search (NAS), which explores the large set of architectural parameters to find the most efficient model to solve a problem. On the contrary, the full potential of the learning complexity metric is deployed in the initial phases of the machine learning pipeline, when deciding how to model the problem to learn. In the particular case of error control, instead of learning the complete error control function, which may result in an unnecessarily large DNN, learning could be solely applied to those specific components that would benefit from it. When designing a search algorithm for which an optimal algorithmic solution already exists (see Sec. 4.3), precisely narrowing down the components to be learned is fundamental. Otherwise, if the resulting DNN required more floating point operations to obtain the HARQ configuration, applying a learning-based approach would not be justified from a sustainability standpoint.

The remainder of this chapter builds on top of the results presented in Chapter 4 in order to reduce the size of the DNN. SHARQ spends most of the time on the optimal repair schedule construction and exploring the complete set of possible block lengths, while p and N_C are efficiently found with a binary search and a closed-form expression, respectively.

Hypothesis 5.1.1 (Adaptive HARQ learning complexity)**.** Predicting all HARQ parameters unnecessarily increases the complexity of the learning problem and hence only the block length should be learned in order to reduce the DNN size.

Solely learning the block length instead of the complete set of parameters (k, p, N_C) would already avoid the most outer loop in Eq. (3.10). The algorithm would go from evaluating k_{lim} configurations in SHARQ to a single one if k is learned.

Introducing DNNs into the search algorithm deviates from the original optimization problem defined in Eq. (3.10): the search no longer outputs the optimal configuration with minimal RI in all cases. A model that perfectly fits the training set with 100% accuracy is likely to achieve poor generalization to unseen inputs due to overfitting. Even in the unlikely event that full accuracy and good generalizability are achieved, the DNN may misclassify an input that is in neither of the datasets. Unlike purely algorithmic solutions, DNNs do not provide classification assurances for every input. However, this sub-optimality could be turned into a feature that reduces the learning complexity of the problem as long as the RI increase is kept within tolerable bounds. Fig. 4.1 shows that small changes in the input parameters produce large changes in the output parameters. This behavior is a direct effect of the quantization of the output, which hinders the clustering of input samples that are close together into a single block length, thereby increasing the learning complexity of the problem. Instead, a smoother solution space could be obtained, e.g., using fewer retransmission cycles or a smaller block length than allowed by D_T despite the RI increase they would entail.

Hypothesis 5.1.2 (Quantization effects)**.** The regularization of the output parameters would avoid the complexity introduced by quantization effects, thereby reducing the resulting model size.

The following section introduces a search algorithm that has been designed following Hyp. 5.1.1 and Hyp. 5.1.2, and provides results that confirm the validity of both hypotheses, thereby showing that reducing the learning complexity of the problem with support for the algorithmic components inherited from SHARQ achieves a lower inference time, which in turn improves the predictability of PRRT's performance guarantees.

5.2 DeepSHARQ: Hybrid Error Coding using Deep Learning

Based on SHARQ's search structure, Deep-learned, Scheduled HARQ (DeepSHARQ) [10] applies learning algorithms to estimate the block length and implements a simple schedule construction to reduce the run-time complexity compared to algorithms that use purely learning and algorithmic solutions.

5.2.1 Design Principles

The SHARQ algorithm has enabled the search for the optimal HARQ configuration in real-time for the first time, mainly due to its three algorithmic optimizations: the reduced computational complexity of the basic performance metrics for the validation of the performance guarantees (see Table 4.1), the Graph Search in Sec. 4.3.2, which reduces the computational complexity of the optimal schedule construction, and the efficient search space exploration in Sec. 4.3.1. Despite these optimizations, the search still fails to provide predictable performance guarantees to the application, especially on embedded devices with limited computational resources, the natural component on CPSs. There are two main contributors to the inference time. Most of the resources are spent on the Graph Search for the $\boldsymbol{pp}$ table construction, which marks a constant lower bound to the inference time, while the full search for all valid block lengths is responsible for the tail latency shown in Sec. 4.4.2. These shortcomings of the SHARQ algorithm are addressed by DeepSHARQ thanks to its two design principles: i) in contrast to purely learning-based approaches, *DeepSHARQ exploits SHARQ's search structure* to simplify the learning problem by applying domain knowledge, which reduces the size of the neural networks, and ii) *DeepSHARQ relaxes the optimality constraint* to provide predictable performance guarantees thanks to a lower and more predictable inference time.

DeepSHARQ reuses from SHARQ the binary search for the optimal p (Alg. 2) and the closed-form expression for the optimal N_C (Eq. (4.8)). For the remaining variables, k and N_P, DeepSHARQ uses a neural network and a simple schedule construction, respectively.

5.2.2 Output Space Regularization

The *universal approximation theorem* shows that neural networks can approach virtually any function with arbitrarily high accuracy, provided they have a sufficient number of parameters [20]. Formally, an adaptive HARQ search algorithm maps the (continuous) channel and application parameters to the (discrete) block length:

$$Adaptive\ HARQ = \{f \mid f : \mathbb{R}^u \rightarrow \mathbb{N}^v\} \tag{5.3}$$

where $u = 6$ is the size of input vector $\boldsymbol{x} = [D_T, PLR_T, T_s, R_C, p_e, RTT]$, and $v = N_C + 2$ accounts for a single output to predict the block length (k), and $N_C + 1$ outputs for the $N_C + 1$ elements in the vector N_P. Although neural networks that predict vectors of arbitrary length exist, an alternative faster construction for the vector N_P is provided in Sec. 5.2.3. A neural network trained to estimate the remaining optimal parameters performs the mapping:

$$NN_{full} = \{f \mid f : \boldsymbol{x} \in \mathbb{R}^6 \rightarrow [k^*, p^*, {N_C}^*] \in [1, 255]^3\} \tag{5.4}$$

where k^*, p^*, and ${N_C}^*$ are the parameters with the minimum RI obtained by SHARQ. Nevertheless, the neural network in DeepSHARQ has been designed to reduce its size so that it can achieve a lower average inference time that would allow it to run on resource-constrained devices. SHARQ's structured search shows that p can be found in logarithmic time, whereas a closed-form expression is available for N_C once the (k, p) tuple is known. Therefore, modeling the complete optimization problem in Sec. 3.5 with deep learning unnecessarily increases the size of the problem to be learned. DeepSHARQ only applies deep learning to the block length inference:

$$NN_{k_{opt}} = \{f \mid f : \boldsymbol{x} \in \mathbb{R}^6 \rightarrow k^* \in [1, 255]\} \tag{5.5}$$

As hypothesized in Sec. 5.1.3, the learning complexity of Eq. (5.5) is high due to the quantization effects depicted in Fig. 4.1. In order to reduce the size of the neural network, DeepSHARQ learns a simpler mapping between input and output space. This is achieved with what we call *output space regularization*, which smoothes the output space as follows: instead of predicting k^*, a set of valid block lengths is defined such that the RI increase is within a certain range of the minimum RI. Formally, given the set $\mathcal{K}_v$ of all block lengths that fulfill the requirements—see Sec. 3.5—, any block length in the set $\mathcal{K}_v^{\Delta} \subset \mathcal{K}_v$ such that $RI(k, N_C, N_P(k)) \leq (1 + \Delta) \cdot RI(k^*, {N_C}^*, {N_P}^*) \, \forall \, k \in \mathcal{K}_v^{\Delta}$ is considered a valid label. Therefore, DeepSHARQ's neural network learns the mapping:

$$NN_{reg} = \{f \mid f : \boldsymbol{x} \in \mathbb{R}^6 \rightarrow \{k \mid \forall \, k \in \mathcal{K}_v^{\Delta}\}\} \tag{5.6}$$

The transition from Eq. (5.4) to Eq. (5.6) has

1. reduced the dimensions of the output, which on its own divides the number of weights in the output layer by three, a significant number given the final model architectures (see Sec. 5.2.4), and

2. simplified the mapping from a single valid label to a set of labels, thereby increasing the probability of smaller neural networks finding a valid class.

The three neural networks have been trained and evaluated in Sec. 5.3.2, which shows that the output space regularization here presented reduces the neural network number of parameters required to achieve 99% accuracy by two orders of magnitude.

Finally, it should be noted that the regularization mechanism here presented differs from the early version of DeepSHARQ presented in [10]. In [10], regularization is only applied to configurations whose block length was limited by the constraints, namely $k^* < 255$. If the channel capacity is enough to meet the application constraints without any redundancy—i.e., $k^* = 1$ and $p^* = 0$—, the neural network was trained to map the input to the minimum block length $k = 1$. The assumption behind this regularization mechanism was that the latter cases should be easily detectable by the neural network due to their straightforward optimal solution, and hence no regularization is required. Nevertheless, later evaluations proved this assumption wrong: higher accuracy is achieved by smaller neural networks when regularization is extended to all cases, and hence this is the regularization that has been finally used for DeepSHARQ.

5.2.3 Repair Schedule Construction

The repair schedule construction is responsible for the long inference times in SHARQ due to the $\boldsymbol{pp}$ table calculation (see Sec. 4.4.2). Two alternative repair schedule constructions are presented in the following: i) an optimal scheduler that does not need to obtain the $\boldsymbol{pp}$ table, and ii) a suboptimal but simple scheduler that follows a fixed parity packet distribution mechanism without the need for RI evaluations. We have finally opted for the simple schedule for DeepSHARQ because Sec. 5.4 shows that it achieves faster inference without a large increase in the transmitted data rate.

Optimal Schedule without $\boldsymbol{pp}$ Table

The objective of the $\boldsymbol{pp}$ table in the Graph Search algorithm (see Alg. 4) is to provide a fast mechanism to evaluate the probability of a cycle failing (see Eq. (4.12)). The full table is required by SHARQ because it explores the complete range of valid block lengths. On the contrary, DeepSHARQ is limited to a single block length, which provides an opportunity to avoid the expensive $\boldsymbol{pp}$ table calculation.

Given $\mathbf{n}[c]$ the number of parity packets transmitted in the c'th repair cycle, Alg. 8 in Appendix B provides a mechanism to obtain the vector $\boldsymbol{w}$, where $\boldsymbol{w}[j]$ is the probability of a repair cycle failing if $\mathbf{n}[c-1] = j$. Formally, $\boldsymbol{w}[\mathbf{n}[c-1]-k] = p_f[c]$. Therefore, the vector $\boldsymbol{w}$ initialized for a fixed k and $j \in [0, p]$ contains all the weights the Graph Search needs to obtain the optimal repair schedule. The substitution of the $\boldsymbol{pp}$ table by the $\boldsymbol{w}$ vector is evaluated in Sec. 5.4, which shows that it still introduces an excessive computational overhead. DeepSHARQ implements a simple schedule construction instead.

Simple Schedule

For every transmission in a binary erasure channel, a p_e'th of the transmitted data gets lost. Therefore, the minimum required redundancy to correct all the erasures in such a channel is:

$$RI_{min} = \left(\sum_{i=0}^{\infty} {p_e}^i\right) - 1 = \frac{p_e}{1 - p_e} \tag{5.7}$$

The long tail for Fast Exploration in Fig. 4.8 suggests that few samples in the dataset really benefit from the faster schedule provided by Graph Search. This section explores a simple schedule construction that can be evaluated in constant run-time complexity, which was originally proposed in [57].

Assuming a time-unbound ARQ scheme is used, the probability of every new cycle being triggered reduces exponentially with every newly transmitted packet. Therefore, retransmitting a single packet per round approaches the RI in Eq. (5.7), and only the feedback in the back channel prevents the channel capacity from being achieved. However, in the packetized transport layer ACKs are usually two orders of magnitude smaller than packets, and hence their overhead is negligible.

Inspired by this theoretically optimum schedule for pure ARQ, DeepSHARQ implements a simple schedule construction. In the time-bound case, a single parity packet per cycle is not enough. If more than one parity packet have not been transmitted before the last cycle, they must all be proactively transmitted. The simple repair schedule is constructed as follows: if $N_C = 0$, then p parity packets are transmitted in the FEC cycle, whereas for $N_C \in [1, p]$ the FEC is kept empty, followed by the all-ones cycles and $p - N_C + 1$ in the last cycle. Despite being suboptimal, such a naive schedule can be constructed in $\mathcal{O}(1)$. Moreover, the results in Sec. 5.4 corroborate the original intuition that few scenarios benefit from the optimal schedule provided by Graph Search.

5.2.4 System Architecture

DeepSHARQ's pipeline is depicted in the central column of Fig. 5.2. The block length is predicted by a neural network that has a configurable number of fully connected hidden layers and neurons, which varies depending on the target predictability level and deviation from the optimum RI. As a rule of thumb, smaller architectures have a low inference time—i.e., are more suitable for low-end embedded devices—at the cost of lower predictability, a higher deviation from the optimum $RI(k^*, N_C, {N_P}^*)$, or both. These trade-offs are further analyzed in Sec. 5.3.2. The activation function in the hidden layers is the ReLU function, while the output layer uses the softmax activation function (see Eq. (5.8)). The activation function takes as input the vector $\mathbf{z} \in \mathbb{R}^K$, where K is the number of classes. The normalization term makes $\sum_{j=1}^{K} softmax(\boldsymbol{z})_i = 1$. Therefore, the neural network outputs can be interpreted as the probability of each block length being the correct one. Fig. 5.3 depicts the smallest neural network architecture evaluated in this chapter, having 3 hidden layers with 10 neurons each. This neural network achieves 99% accuracy for $\Delta = \infty$, and hence it is the lower bound of how small the

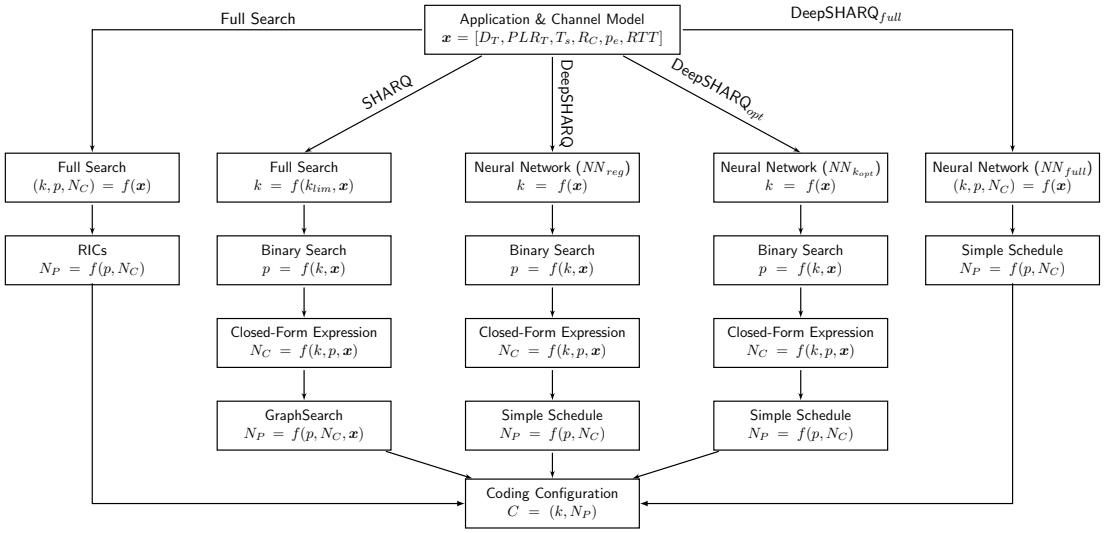


Figure 5.2: Architecture and Information Flow for the different Search Algorithms. Each column represents one algorithm implementation. The vector $\boldsymbol{x}$ includes the application and channel model

models obtained with the DeepSHARQ methodology here presented can be while still achieving high accuracy. Whether the number of parameters, and hence inference time, can be further reduced with other techniques such as weight quantization [115, 141] or weight pruning [118, 133] is left for future work. In the remainder of this chapter, we constrain ourselves to evaluate the performance achieved by DeepSHARQ alone.

$$softmax(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{5.8}$$

The binary search for p and the closed-form expression for N_C are taken from the SHARQ algorithm (see Sec. 4.3). Finally, the graph search in Sec. 4.3.2 has been substituted by a simpler repair schedule construction (see Sec. 5.2.3). The block length estimation, the repair schedule construction, and obtaining the optimal number of repair cycles have run-time complexity $\mathcal{O}(1)$. Therefore, the major contributors to DeepSHARQ's complexity are the binary search for p and the RI calculation to ensure that the requirements are fulfilled. According to Table 4.1, the RI calculation has a computational complexity $\mathcal{O}(k_{lim} + p_{opt})$. The binary search evaluates Eq. (3.17) at maximum m times, where m is the number of bits per symbol in the employed Galois Field $GF(2^m)$—see Sec. 3.1.2. The binary search has a computational complexity $\mathcal{O}\big(m \cdot (k_{lim} + log(p_{opt}))\big)$—see Table 4.1 for the complexity of the PLR function. In the transport layer, $m = 8$ so that symbols are one-byte long and $k_{max} = p_{max} = 2^m$,[3] resulting in DeepSHARQ's run-time complexity $\mathcal{O}(m \cdot k_{max} + p_{max})$.

[3]Given a systematic code in $GF(2^m)$, it is theoretically possible to construct a $\mathcal{C}(k_{max}, p_{max})$ code with $k_{max} = p_{max} = 2^m$. However, the MDS coder implementation considered in this paper enforces

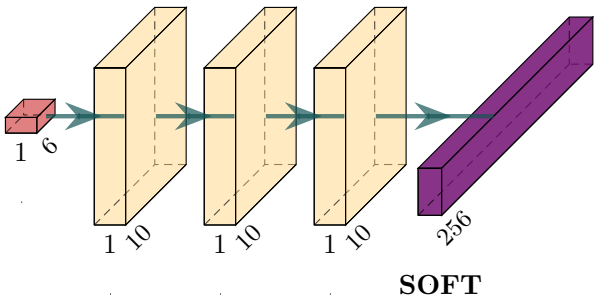


Figure 5.3: DeepSHARQ's neural network architecture.

Fig. 5.2 shows other two versions of DeepSHARQ whose main difference is the implemented neural network. DeepSHARQ$_{opt}$'s neural network estimates the optimal block length—i.e., it implements Eq. (5.5)—, whereas DeepSHARQ$_{full}$ implements Eq. (5.4) to estimate the three parameters k, p, and N_C. These algorithms are only provided to prove Hyp. 5.1.1 and Hyp. 5.1.2 and the PRRT protocol does not support any of them.

5.2.5 Loss Function

Given a discrete random variable X taking values in an alphabet $\mathcal{X}$, the *cross-entropy* between two probability distributions $p_x : \mathcal{X} \rightarrow [0, 1]$ and $q_x : \mathcal{X} \rightarrow [0, 1]$ is:

$$H(p_x, q_x) = -\sum_{x \in \mathcal{X}} p_x(x) \cdot log(q_x(x)) \tag{5.9}$$

Since the cross-entropy is a measure of the difference between the distributions p_x and q_x, it has been frequently used in the context of classification problems as a loss function, where p_x is the probability distribution of the true labels and q_x is the probability distribution predicted by the neural network. Provided one-hot coding is used for the target vector—i.e., $p_x(x) = 0 \; \forall \; x \in \mathcal{X} \setminus x_{true}$ and $p_x(x_{true}) = 1$, where x_{true} is the true class—, Eq. (5.9) boils down to what is known as the *categorical cross-entropy*:

$$H(p_x, q_x) = -\sum_{x \in \mathcal{X}} p_x(x) \cdot log(q_x(x)) = -log(q_x(x_{true})) \tag{5.10}$$

The categorical cross-entropy loss can be used with an arbitrary number of classes. However, if $X = \{x_1, x_2\}$ is a binary random variable, Eq. (5.9) boils down to the *binary cross-entropy*:

$$H(p_x, q_x) = -p_x(x_1) \cdot log(q_x(x_1)) - (1 - p_x(x_1)) \cdot log(1 - q_x(x_1)) \tag{5.11}$$

the code (k, p) to fulfill that $k + p \leq 2^m$. Therefore, both variables must be independently treated in the complexity analysis.

In the particular case of DeepSHARQ, the problem combines multi-class and binary classification. Multi-class because the neural network predicts one out of 256 classes—i.e., block lengths with $k = 0$ denoting there is no valid configuration that meets the constraints, see Sec. 3.5.1. Binary because the model training is guided by a new binary cross-entropy loss that considers the probability of predicting a block length belonging to one of the two sets—i.e., the sets of valid and invalid block lengths. The random variable $X : [0, 255] \rightarrow \{1, -1\}$ is defined to group the block lengths into the valid and invalid set with the mappings $k \in K_v^\Delta \rightarrow 1$ and $k \notin K_v^\Delta \rightarrow -1$. Therefore, the probability distribution functions in the binary cross-entropy loss are $p_x(x)$ the probability that the true label belongs to the valid set (see Eq. (5.12)), and $q_x(x)$ the probability of picking any block length that belongs to the valid set predicted by the neural network (see Eq. (5.13)).

$$p(x) = Prob[k \in K_v^\Delta] = \begin{cases} 1 & \text{if } x = 1 \\ 0 & \text{if } x = -1 \end{cases} \tag{5.12}$$

$$q(x) = Prob[\hat{k} \in K_v^\Delta] = \begin{cases} \sum_{k \in K_v^\Delta} softmax(\mathbf{z})_k & \text{if } x = 1 \\ \sum_{k \notin K_v^\Delta} softmax(\mathbf{z})_k & \text{if } x = -1 \end{cases} \tag{5.13}$$

The major disadvantage of such a loss definition is that it operates with tensors of different sizes, as the range $[k_{min}, k_{max}]$ may differ between any two inputs. Therefore, unlike the cross-entropy loss, DeepSHARQ's loss must be evaluated on a per-sample basis instead of a per-batch basis, which introduces a non-negligible overhead in the training phase. The following section analyzes how the models have been trained, including an analysis of such a drawback.

5.3 Model Training

Both the dataset generation and the ablation study are central to any deep learning pipeline. The dataset should be generated such that it represents the conditions the model will face once deployed. The ablation study ensures that the selected neural network generalizes to unseen samples and it is not larger than required to achieve high prediction accuracy. How these two steps have been realized for DeepSHARQ is presented in the following.

5.3.1 Dataset Generation

Carefully designing a dataset generation methodology is instrumental for any machine learning project. The model's ability to generalize to unseen inputs and its performance under real conditions depend on it. With this in mind, the dataset should fulfill two requirements:

Requirement 5.3.1 (Faithfulness)**.** Failing to faithfully represent network conditions may go unnoticed in the training phase if the model achieves high accuracy in all datasets

and poor performance will only be experienced once the model is deployed. Therefore, the dataset must represent realistic conditions that the protocol will experience once deployed in real networks.

Requirement 5.3.2 (Generalizability)**.** Small datasets are likely to misrepresent the conditions that the model encounters in practice. This lack of generalizability is visible when the model overfits the training dataset while performing poorly in the validation dataset. Therefore, the dataset must be large enough so that the training and validation losses are not too far apart.

All the models have the same 6 input parameters:

- Application parameters: target delay (D_T), packet loss rate (PLR_T), and source packet interval (T_s).
- Network parameters: channel erasure rate (p_e), round-trip time (RTT), and bottleneck data rate (R_C).

Throughput, round-trip time and loss rate traces are widely available for the most common network deployments nowadays—i.e., broadband [126], 4G [145], 5G [178, 184], and WiFi [120, 131]. When it comes to application parameters, the dataset includes delay and reliability constraints of traditional applications, e.g., real-time video and audio [77], as well as more demanding application, e.g., the Tactile Internet [101] or Virtual Reality [125]. The source packet interval of multimedia applications [94] as well as control applications [136] has also been considered. Having been generated with the aforementioned network and application parameters, the dataset in Sec. 3.6 fulfills Req. 5.3.1. The results presented in Sec. 5.3.2 show that it fulfills Req. 5.3.2 as well. The dataset has been extended so that not only the optimal block length is logged, but DeepSHARQ needs that k_{min} and k_{max} are also obtained, which are the minimum and maximum block lengths that ensure the RI only deviates Δ from the optimum (see Sec. 5.2.2).

The delay and loss rate models in Sec. 3.3 consider other three parameters that have not been included in the dataset here presented. The packet length (L_p) is assumed to be fixed to the MTU of the underlying physical medium, which in IP networks is typically limited by the Ethernet MTU (i.e., $L_p = 1,500$ bytes). Although precise processing delay information could be fed into the algorithm [3, 6], doing so would increase the dimensions of the dataset and likely the complexity of the resulting neural networks as well. Therefore, a rather conservative constant processing delay $D_{RS} = 1$ ms has been considered. Finally, the packet loss detection delay (D_{PL}) is linearly dependent on the source packet interval, and hence it adds no new information the neural network can learn from.

5.3.2 Ablation Study

Tuning the learning rate hyperparameter is instrumental to successfully training neural networks. An ablation study consists of the removal of different components of the system in order to better understand and analyze their impact on the performance of

the complete system. The different hyperparameters can be fine-tuned based on such a study, thereby improving the training methodology and model performance.

Models trained with a constant learning rate showed poor performance from the beginning of the ablation study. While constant high learning rates fail to converge, low learning rates converge very slowly, resulting in long training sessions. PyTorch implements various dynamic learning rates policies, such as *super-convergence*[4] [164] or *plateau*[5]. Both policies combine a high learning rate phase for solution space exploration with a low learning rate phase to fine-tune the model parameters. The plateau learning rate policy monitors the validation loss to estimate the effectiveness of the current learning rate: if after *patience* epochs the validation loss did not decrease by at least $threshold$ amount, it decays the learning rate by a constant $factor$ until the min_lr has been reached. Super-convergence begins with a rising phase that goes from $start\_lr$ to max_lr, after which it decays towards end_lr a substantially lower learning rate than $start\_lr$. The super-convergence policy requires an optimizer with momentum, and hence we trained all the models with momentum-enabled stochastic gradient descent. Super-convergence varies the momentum between 0.85 and 0.95, while it is constant at 0.9 for the plateau policy.

Fig. 5.4 shows the accuracy and learning rate evolution for models trained for 150 epochs (more on why this is the selected number of epochs later) with both policies. The configured patience and reduction factors in the plateau policy result in opposed learning rate schedules: the fastest convergence towards low learning rates is achieved by *Plateau (0.4,5)*, while the slow adaptation in *Plateau (0.8,10)* makes the schedule stay in high learning rates throughout the complete training. Nevertheless, super-convergence outperforms plateau in all the evaluated scenarios due to its extended high learning rate exploration phase. Super-convergence with $max\_lr = 0.04$ has been the selected learning rate policy to train the models for the remainder of this section, as it achieves a better performance than lower maximum learning rates (see 0.02 and 0.03) and further increasing the learning rate does not increase the accuracy (see 0.05).

The model performance comparison is based on two different metrics. On the one hand, the accuracy measures the percentage of correctly predicted samples, and hence it measures a model's performance from a learning point of view. On the other hand, the percentage of valid configurations measures the ability of the neural network to predict a block length that meets all the constraints independently of the produced RI overhead, and hence it measures the model performance from an information-theoretical standpoint. Unless stated otherwise, the values for these two metrics presented in the following have been obtained with the test dataset. All models presented in the following have been trained with PyTorch 1.12.1.

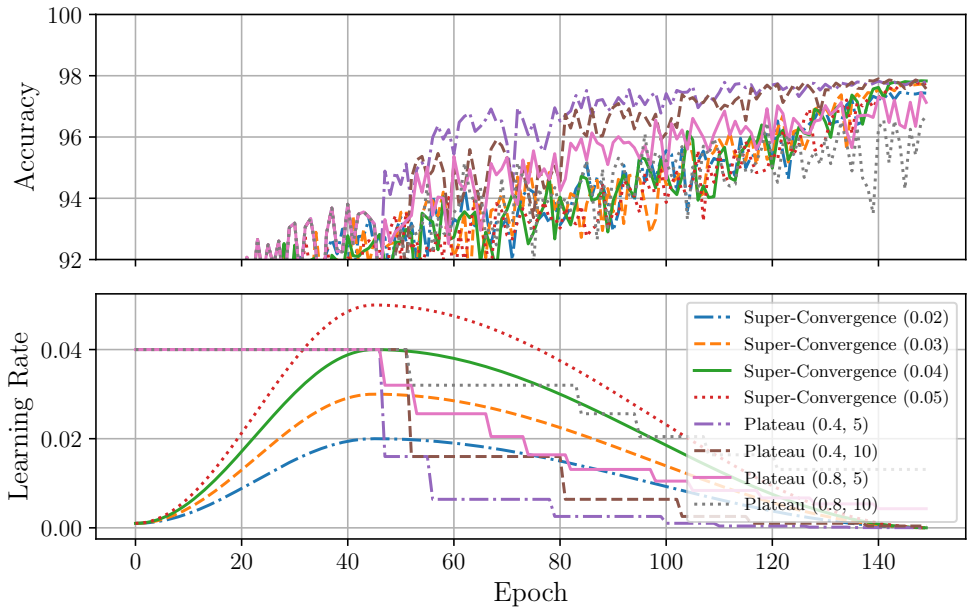


Figure 5.4: Validation accuracy and learning rate evolution for different learning rate policies. Three maximum learning rates have been used for super-convergence (i.e., 0.02, 0.03, 0.04, and 0.05), whereas two different patience (5 and 10) and learning rate reduction factors (0.4 and 0.8) have been used for plateau.

DeepSHARQ

The number of epochs to train the models has been selected based on the results presented in Table 5.1. Assuming the regularization factor is fixed to 10^{-5}, the model trained for 150 epochs outperforms for more than 1% the model trained for 50 epochs. Although increasing the epochs to 250 further increases the model's accuracy, these improvements become asymptotically small as the training time keeps increasing. On an Intel Xeon E3-1241 v3 CPU at 3.5GHz and 8 cores, training for 150 and 250 epochs results in training sessions of 4-5 and 6-7 hours, respectively. The training times reported here are the observed upper and lower bounds for the models presented later in Table 5.2, which vary with the number of parameters in the model. Training the models for two additional hours for marginal accuracy improvements is not justified from a sustainability standpoint (see Sec. 5.1.2) and hence 150 epochs are used in the following.

Table 5.1 also compares different factors for L_2 regularization. The high learning rates in the super-convergence policy already introduce some regularization in the model [164]. Therefore, a high regularization factor such as 10^{-4} prevents the model from achieving

[4] `https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.OneCycleLR.html` (accessed January 23rd 2024)

[5] `https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html` (accessed January 23rd 2024)

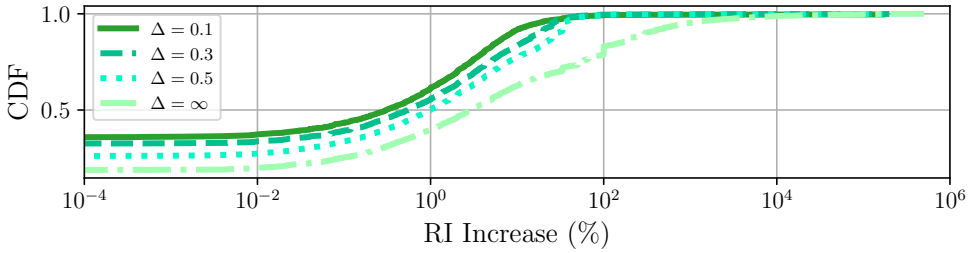
Table 5.1: Effects of the number of epochs and regularization factor on the model accuracy and the number of valid configurations. All the models have been trained for $\Delta = 0.3$ and their architecture is 3 hidden layers and 20 neurons per layer.

Epochs	Regularization	Accuracy	Valid Configurations
50	10^{-5}	96.69%	98.18%
150	10^{-4}	96.14%	97.86%
150	10^{-5}	97.80%	98.90%
150	10^{-6}	97.96%	98.99%
250	10^{-5}	97.80%	98.78%

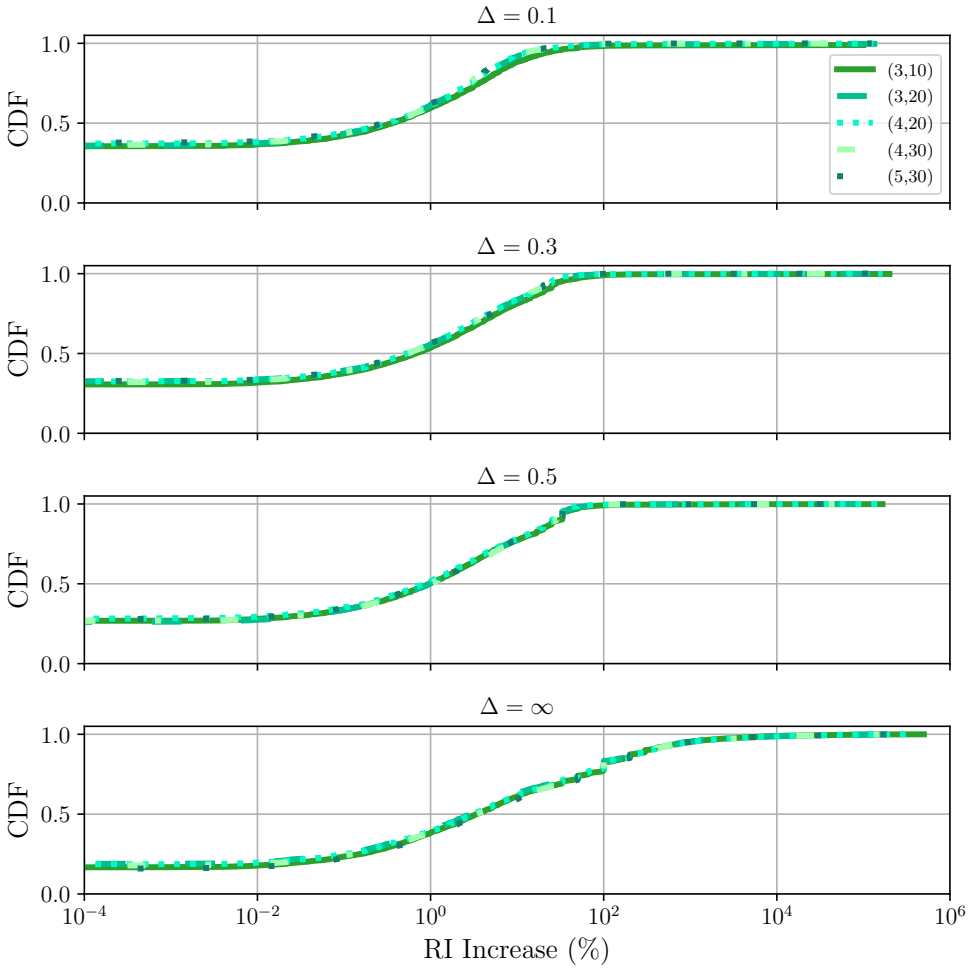
enough complexity for highly accurate predictions. On the contrary, low regularization factors such as 10^{-5} and 10^{-6} do increase the model accuracy. Although 10^{-6} models outperform 10^{-5} from a pure learning perspective—i.e., higher accuracy—and an information theoretical point of view—i.e., lower percentage of valid configurations—, models with a lower number of parameters trained for such a low regularization factor experience exploding gradient. As larger models are more prone to overfitting, since they can learn more complex mathematical functions, we have finally opted for the more conservative 10^{-5} to prevent overfitting in these cases.

The accuracy and number of valid configurations for different neural network architectures and $\Delta = [0.1, 0.3, 0.5, \infty]$ are collected in Table 5.2. As expected, larger Δ's allow smaller models to achieve high accuracy and percentage of valid configurations due to the reduced complexity of the problem to learn. When the models are allowed to learn any valid configuration—i.e., $\Delta = \infty$—, all the trained models achieve 99% accuracy. These models set the upper bound to the achievable accuracy as the models become smaller for the output regularization mechanism with the training methodology here presented. On the other end of the spectrum are the models trained with $\Delta = 0.1$, which fail to achieve high accuracy with all network architectures. As the learning complexity increases, larger and more computationally expensive neural networks are required to learn the problem with high accuracy.

To better understand the relation between network and learning complexity, Fig. 5.5 shows the RI increase from the optimum introduced by every model. The RI deviation from the optimum in DeepSHARQ can come either from misclassifications that still meet the constraints...i.e., valid configurations—, or the simple repair schedule construction. Fig. 5.5a shows a clear performance gap between $\Delta \in [0.1, 0.3, 0.5]$ and $\Delta = \infty$ for a fixed neural network architecture. However, for a fixed Δ, Fig. 5.5 shows the neural network size has little impact on the RI increase distribution, despite the clear differences in terms of accuracy and number of valid configurations (see Table 5.2). The median inference time for the different NN architectures is depicted in Fig. 5.6. The depicted results have been obtained for the samples in the test dataset executed on the same desktop PC and Raspberry Pi Zero W described in Sec. 4.4.1. As expected, the predictability of the inference time is high across all the samples in the test dataset. In order to support as many resource-constrained devices as possible due to its low inference time, the model



(a) RI increase by Δ for $(3, 20)$ neural network.



(b) RI increase by neural network architecture for different Δs.

Figure 5.5: RI increase by neural network architecture and Δ.

Table 5.2: Accuracy and percentage of valid configurations for DeepSHARQ's neural networks trained for 150 epochs with L_2 regularization with a factor of 10^{-5}.

Parameters	Layers	Neurons	Δ	Accuracy	Valid Configurations	Train Time
3,106	3	10	0.1	90.09%	96.25%	285 min
3,106	3	10	0.3	96.31%	98.34%	299 min
3,106	3	10	0.5	97.83%	98.60%	304 min
3,106	3	10	∞	99.01%	99.01%	322 min
6,356	3	20	0.1	94.90%	98.10%	288 min
6,356	3	20	0.3	97.80%	98.90%	302 min
6,356	3	20	0.5	98.60%	99.05%	306 min
6,356	3	20	∞	99.17%	99.17%	364 min
6,776	4	20	0.1	95.63%	98.26%	313 min
6,776	4	20	0.3	97.80%	98.68%	319 min
6,776	4	20	0.5	98.95%	99.20%	335 min
6,776	4	20	∞	99.28%	99.28%	353 min
10,936	4	30	0.1	96.36%	98.43%	309 min
10,936	4	30	0.3	98.52%	99.09%	304 min
10,936	4	30	0.5	99.07%	99.27%	309 min
10,936	4	30	∞	99.30%	99.30%	329 min
11,866	5	30	0.1	96.91%	98.55%	302 min
11,866	5	30	0.3	98.40%	98.91%	315 min
11,866	5	30	0.5	99.19%	99.33%	309 min
11,866	5	30	∞	99.38%	99.38%	328 min

with 3 hidden layers and 10 neurons per layer has been selected as the default model for DeepSHARQ. Nevertheless, the methodology here presented could be used to redesign the neural network to better fit the particular needs of other devices for which such a neural network may still be too large.

DeepSHARQ$_{opt}$

Unlike DeepSHARQ, the neural network in DeepSHARQ$_{opt}$ is trained to predict the optimal block length k^*—i.e., the block length minimizing the RI as obtained by the SHARQ algorithm. The results of the ablation study for DeepSHARQ$_{opt}$ are presented in Table 5.3. The largest evaluated neural network for DeepSHARQ has 5 hidden layers with 30 neurons and was trained for 150 epochs (see Table 5.2). When the same neural network is trained to predict k^* instead, not only does it achieve lower accuracy, but a lower percentage of valid configurations as well. Increasing the number of epochs to 1,000 improves the model accuracy, but longer training sessions must be accompanied by larger neural networks in order to achieve high accuracy. While DeepSHARQ only needs neural networks with 3,106 parameters to achieve 99.86% valid configurations for $\Delta = \infty$, DeepSHARQ$_{opt}$'s neural networks require at least 213,656 parameters to do so.

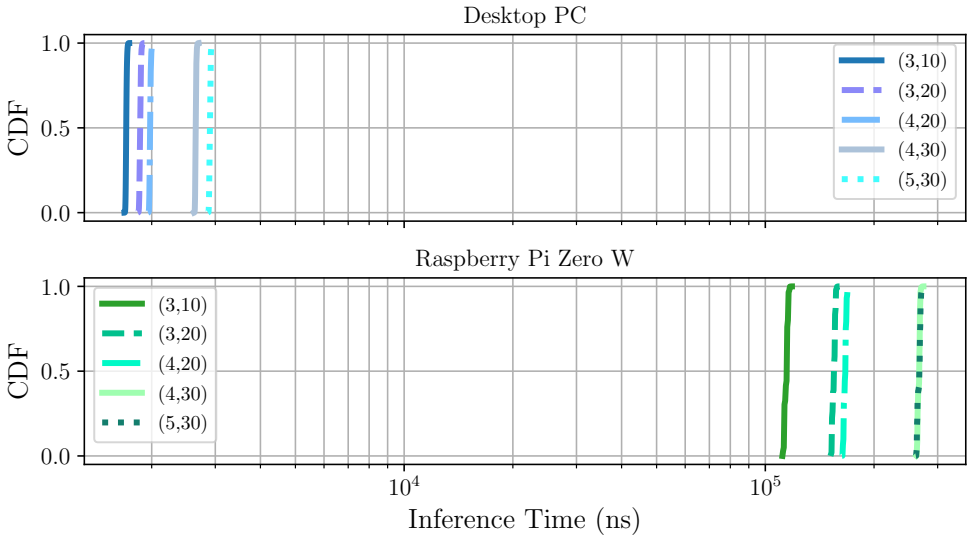


Figure 5.6: CDF of the median inference time for different neural network architectures on the Desktop PC and the Raspberry Pi Zero W.

Table 5.3: Accuracy and percentage of valid configurations for DeepSHARQ$_{opt}$'s neural networks trained with L_2 regularization with a factor of 10^{-5}.

Parameters	Layers	Neurons	Epochs	Accuracy	Valid Configs	Train Time
11,866	5	30	150	72.64%	94.36%	14 min
11,866	5	30	1,000	77.45%	93.17%	93 min
130,306	5	150	1,000	98.81%	99.68%	157 min
213,656	5	200	1,000	99.72%	99.86%	178 min
254,256	4	250	1,000	99.66%	99.85%	204 min
317,006	5	250	1,000	99.83%	99.91%	229 min

On the other hand, as DeepSHARQ$_{opt}$ is trained with the cross-entropy loss, the models are trained significantly faster than DeepSHARQ's even when they have several orders of magnitude more parameters and are trained for 1,000 epochs. Comparing again the models with 99.86% valid configurations, (5,150) DeepSHARQ$_{opt}$ takes approximately two hours less than (5,30) DeepSHARQ with $\Delta = \infty$.

DeepSHARQ$_{full}$

As the learning complexity of the problem increases, so does the neural network size required to cover a higher percentage of valid configurations. DeepSHARQ$_{full}$ predicts the optimal parameters k^*, p^*, and $N_C{}^*$, hence its output layer has 768 neurons, which are split into three groups of 256 for the classification of each of the HARQ parameters. An early version of this algorithm has been published in [9] with a different repair

Table 5.4: Accuracy and percentage of valid configurations for DeepSHARQ$_{full}$'s neural networks trained for 1,000 epochs with L_2 regularization with a factor of 10^{-5}.

Parameters	Layers	Neurons	Accuracy			Valid Configurations
			k	p	N_C	
162,318	3	150	92.71%	91.64%	99.42%	95.45%
184,968	4	150	96.34%	96.02%	99.76%	97.64%
276,368	4	200	97.87%	97.50%	99.87%	98.52%
316,568	5	200	98.95%	98.84%	99.95%	99.26%
382,768	4	250	98.67%	98.49%	99.94%	99.03%
445,518	5	250	99.63%	99.61%	99.97%	99.7%

Figure 5.7: Model conversion pipeline.

schedule construction, which is not presented here as it introduces a high computational complexity without achieving significant performance improvements over the simple repair schedule construction in Sec. 5.2.3. Table 5.4 shows the results of the ablation study. DeepSHARQ$_{full}$ requires 4.3× and 143× more parameters than DeepSHARQ$_{opt}$ and DeepSHARQ, respectively, to achieve 99% valid configurations.

Despite using the same loss function as DeepSHARQ$_{opt}$, DeepSHARQ$_{full}$ models take longer to train due to the high number of neural network parameters. The fastest training session took 339 minutes for the model with 5 hidden layers and 200 neurons per layer, whereas the slowest one took 451 minutes for the model with 4 hidden layers and 250 neurons per layer.

5.4 Performance Analysis

Once the neural network architecture has been decided based on the results of the ablation study, the performance of the new proposed approach is evaluated and compared to previous algorithms.

5.4.1 Methodology

The methodology that has been followed to evaluate the performance of DeepSHARQ algorithm is the same as in Sec. 4.4.2 for SHARQ. The same platforms have been used to carry out the evaluations: a desktop PC running Ubuntu 22.04.2 LTS with Linux kernel 5.19 on an Intel Core i7-7700 CPU at 3.6 GHz, and a Raspberry Pi Zero W running the Raspbian Buster operating system with Linux Kernel 4.19. The predictability of the performance guarantees and the inference time are again the metrics used to compare the performance of the algorithms.

Although PyTorch is a powerful tool for the research of deep learning models, it is not suited for production due to its slow inference and its large binary size. The two major frameworks for the deployment of neural networks on edge devices are PyTorch Mobile[6] and TensorFlow Lite[7], which are the production-ready counterparts to PyTorch and TensorFlow, respectively. While the networks have been trained in PyTorch due to its flexibility, which allows to re-design of almost every step in the learning pipeline, TensorFlow Lite has been selected over PyTorch Mobile because it achieves a faster execution of fully connected models. The `tflite`[8] crate has been used to integrate neural network execution into Rust, which provides Rust wrappers for TensorFlow Lite. The model conversion pipeline depicted in Fig. 5.7 has been implemented to convert PyTorch models into TensorFlow Lite models. PyTorch can save models with the Open Neural Network Exchange (ONNX) format, which is an open standard for representing machine learning models. The TensorFlow Backend for ONNX converts ONNX models into TensorFlow models. Finally, TensorFlow converts the model into the lite format used by TensorFlow Lite. Implementation differences between PyTorch and TensorFlow introduce variations in the output from the originally trained model. However, the networks are robust enough to make these variations negligible. For example, all models trained for $\Delta = \infty$ predict the same output with PyTorch (model.torch) and TensorFlow Lite (model.tflite) for the complete test dataset, but this flawless conversion does not necessarily generalize for every model.

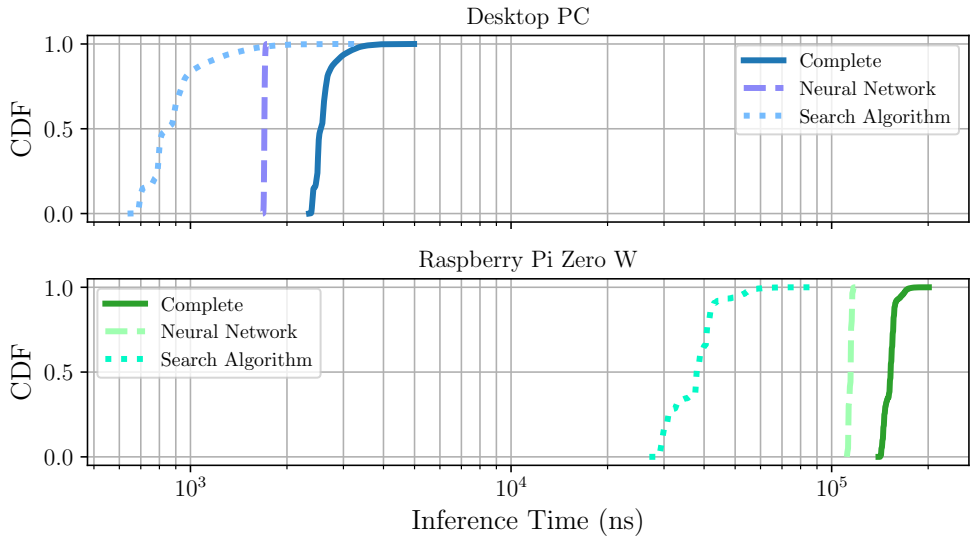
5.4.2 Evaluation

This section evaluates DeepSHARQ's inference time and channel predictability guarantees, comparing it to other search algorithms, and analyzes its underlying design principles and their impact on the overall protocol performance.

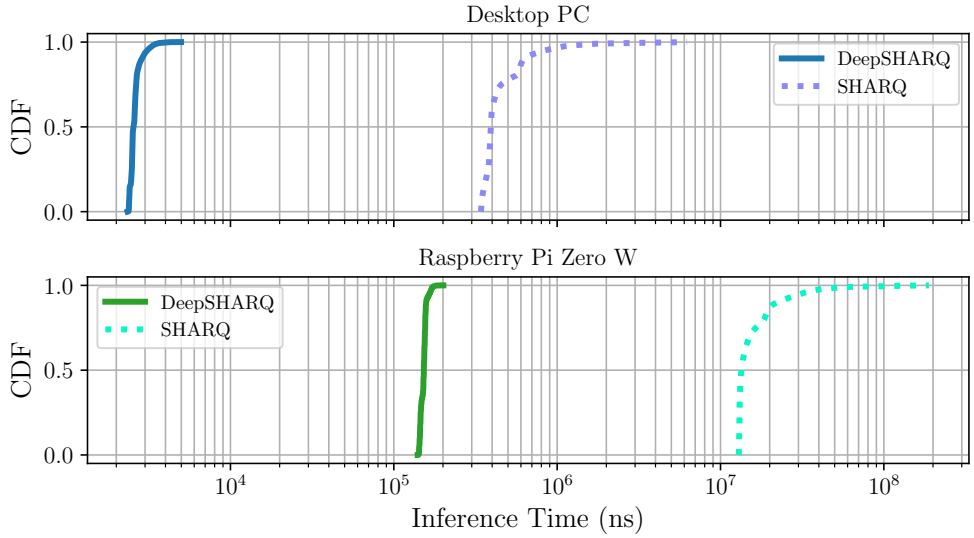
Inference Time

The median inference time of the complete algorithm is depicted in Fig. 5.8a together with the split between NN and algorithmic search. In both platforms, the neural network takes longer to execute than the search components, thereby resulting in a more predictable inference time than previous algorithms (see Sec. 4.4). The difference between NN and search inference time is shorter on the PC than on the Pi, most likely because its CPU provides instructions that better support NN inference—e.g., Single Instruction/Multiple Data (SIMD) instructions. As a result, the CDF of DeepSHARQ's inference time has a longer tail on the PC. Although smaller NNs may be required to support more constrained devices, they would result in lower predictability as the search components dominate the inference time.

[6] `https://pytorch.org/mobile/home/` (accessed January 23rd 2024)
[7] `https://www.tensorflow.org/lite` (accessed January 23rd 2024)
[8] `https://crates.io/crates/tflite/0.1.0` (accessed January 23rd 2024)



(a) CDF of the median inference time split by neural network's and search algorithm's contribution to the complete algorithm



(b) Inference time comparison between DeepSHARQ and SHARQ.

Figure 5.8: DeepSHARQ inference time analysis.

Table 5.5: DeepSHARQ percentage of inferences on the test dataset meeting the ϵ-fulfilling requirement.

Device	ϵ					
	0.75	**0.5**	**0.25**	**0.1**	**0.05**	**0.01**
PC	100%	100%	100%	100%	100%	100%
Pi	100%	100%	100%	100%	99.9%	92.21%

DeepSHARQ's inference time its two orders of magnitude smaller than SHARQ's (see Fig. 5.8b). Moreover, DeepSHARQ presents a more predictable inference time with a tail latency that is in the same order of magnitude as the average inference time.

Communication Channel Predictability

When running on the Raspberry Pi Zero W, DeepSHARQ achieves sub-millisecond inference time, thereby bringing a fast reaction to channel changes to resource-constrained devices for the first time. DeepSHARQ meets all the performance predictability requirements defined in Sec. 4.1 for every single sample in the dataset and on both platforms, the desktop PC and the Raspberry Pi Zero W. In addition, Table 5.5 shows that DeepSHARQ can react to every channel change that occurs within the first 99% of the time budget—i.e., it is 0.01-fulfilling—when executed on a powerful desktop PC. On the Pi, DeepSHARQ reacts to channel changes occurring 5% and 1% before the end of the deadline in 99.9% and 92.21% of the dataset samples, which is a significantly higher performance predictability than SHARQ's (see Sec. 4.4.2).

Optimality

As DeepSHARQ neither predicts the optimal block length nor uses the optimal repair schedule construction, the remainder of this section analyses the impact of such a design decision on the inference time and the RI. DeepSHARQ (3,10)—i.e., 3 hidden layers with 10 neurons each—and $\Delta = \infty$ is compared to DeepSHARQ$_{opt}$ (5,150). The neural network of DeepSHARQ$_{opt}$ has been selected because it is the smallest one that achieves more than 99% of valid configurations. The two neural networks have been evaluated for three repair schedule policies: i) the simple schedule in Sec. 5.2.3, ii) the optimal schedule without ***pp*** table in Sec. 5.2.3, and iii) the optimal schedule with ***pp*** table in Sec. 4.3.2.

Fig. 5.9 compares the inference time of the six configurations. When the ***pp*** table is used to construct the optimal schedule, the inference time with both neural networks is very similar because the ***pp*** table construction dominates the inference time and takes one or two orders of magnitude longer than the NN inference. Without the ***pp*** table, the inference time is in the same order of magnitude as the inference of the (5,150) neural network, which explains the more perceivable difference between DeepSHARQ and DeepSHARQ$_{opt}$. Finally, thanks to the reduced size of its neural network, DeepSHARQ with the simple schedule achieves the lowest inference time of the six configurations.

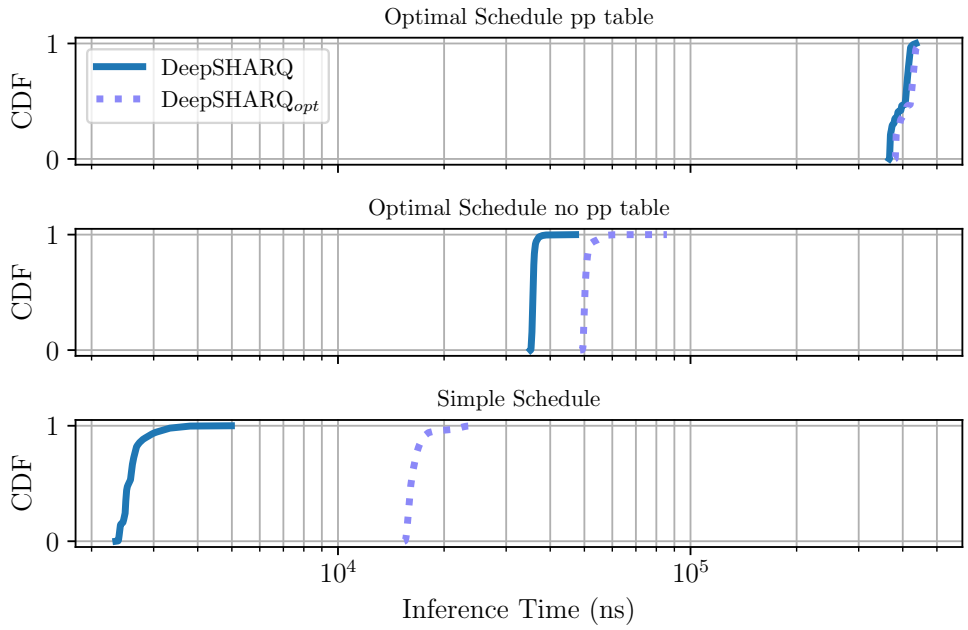


Figure 5.9: DeepSHARQ and DeepSHARQ$_{opt}$ inference time on the desktop PC with the optimal and simple schedule constructions.

On the one hand, the inference time provides a measure of the CPU resources that are devoted to obtaining the optimal block length and repair schedule. On the other hand, the data rate increase with respect to the minimum data rate achieved by SHARQ provides a measure of the communication resources that are wasted when configuring a suboptimal block length and repair schedule. Fig. 5.10 depicts the results for the latter. The difference between DeepSHARQ and DeepSHARQ$_{opt}$ is clear at first sight, which stems from the neural network design—i.e., whether it is trained to predict the optimal block length or not. Nevertheless, the upper quartile and the maximum data rate increase are 0.2% and 2% higher than DeepSHARQ$_{opt}$'s, respectively. DeepSHARQ$_{opt}$ achieves the same data rate with the simple and optimal schedules in 98.44% of the cases, whereas DeepSHARQ does so in 98.16% of the cases. Therefore, the inference time increase does not seem justified for a schedule difference that only affects less than 2% of the samples in the dataset, especially for DeepSHARQ given its low inference time.

5.5 Discussion

At the core of DeepSHARQ's design principles is the deviation from the original optimization problem, namely finding the HARQ configuration with minimum RI as SHARQ does. This section discusses the trade-offs of these two opposed search algorithm designs.

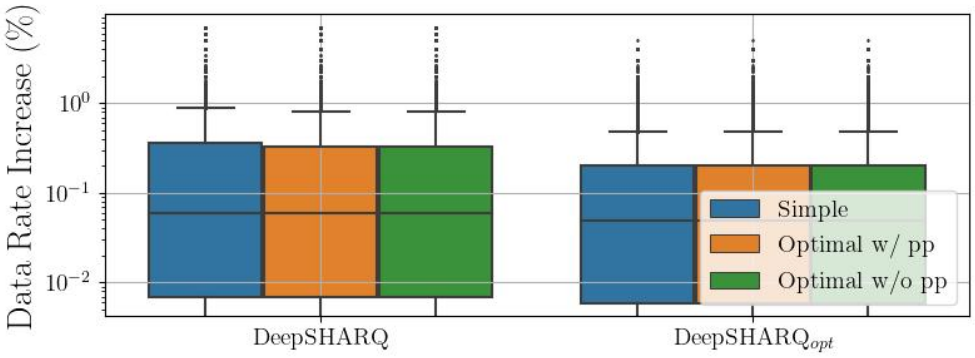


Figure 5.10: DeepSHARQ and DeepSHARQ$_{opt}$ data rate increase with the optimal and simple schedule constructions.

5.5.1 The Cost of Optimality

DeepSHARQ introduces two sources of suboptimal HARQ configurations. Firstly, the neural networks can experience misclassifications even when trained to predict the optimum, but the regularization mechanism presented in this chapter even encourages suboptimal block lengths in order to reduce the learning complexity of the problem. Secondly, the simple schedule avoids the computationally expensive Graph Search with a suboptimal schedule construction inspired by pure ARQ retransmissions.

DeepSHARQ does not only predict suboptimal HARQ configurations, it provides *predictably suboptimal* HARQ configurations. The parameter Δ controls the RI overhead that the predicted block lengths can introduce. However, Δ also controls the learning complexity of the problem and thus the trade-off between CPU and network resources (see Sec. 5.4 for more details on the trade-off). Small Δ's approach the optimal HARQ configuration more closely, which in turn increases the inference time due to the larger neural network required to learn the problem. Conversely, large Δ's accept more block lengths as valid, thereby enabling smaller neural networks to support a high percentage of valid configurations. On the other hand, the simple schedule construction further reduces the computational complexity of the algorithm at the cost of suboptimal configurations. Sec. 5.4 shows that aiming at the optimal configuration with DeepSHARQ increases the inference time by an order of magnitude but has negligible impact on the transmitted RI. Devoting so many resources to finding the optimum when few configurations really benefit from it defeats one of the main design targets of DeepSHARQ—i.e., the sustainable deployment on embedded systems discussed in Sec. 5.1.2. Until a more efficient search algorithm is found that solves Eq. (3.10) with fewer resources than SHARQ, adaptive HARQ on resource-constrained devices will be limited to suboptimal algorithms such as DeepSHARQ.

5.5.2 The Cost of Suboptimality

The downside of DeepSHARQ's suboptimality is the long training sessions, despite training smaller neural networks. Such an increase in the training time should be analyzed from a sustainability standpoint. DeepSHARQ uses a modified version of the binary cross-entropy (see Sec. 5.2.5), which instead of taking a single valid label as the traditional binary cross-entropy, considers a range of valid labels. The impact of such a loss function in the training phase is two-fold: the number of epochs to achieve a high accuracy is reduced by an order of magnitude—in Sec. 5.3.2 we show that it goes from 1,000 to 150 epochs—but epochs take significantly longer to be processed. These results suggest that the proposed output regularization is better suited for TinyML than it is for large models because the longer training could be compensated for with more efficient inference once the model is deployed.

For the particular case of DeepSHARQ, Sec. 5.3.2 shows that its training sessions take longer but are not far from models learning the optimal block length. DeepSHARQ has enabled predictable performance guarantees on resource-constrained devices for the first time, thereby increasing its chances of being deployed at scale. Once deployed at scale, the longer training should be quickly compensated with a more energy-efficient reaction to channel changes than if DeepSHARQ$_{opt}$ models would be used instead. Moreover, embedded systems are usually powered by batteries so a more energy-efficient operation increases their operation lifetime.

A final aspect to consider is the free parameter Δ. The neural network will have to be retrained in order for DeepSHARQ to accommodate applications with different reliability requirements. Transfer learning [46] could be used to quickly adapt pre-trained models to new Δ configurations without going through the complete learning cycle.

5.5.3 Striking the Right Balance

DeepSHARQ learns to find the right balance between proactive and reactive redundancy subject to reliability, delay, and data rate restrictions. However, its design consisted of finding yet another balance: that of addressing the different components of the system with either algorithmic or learning solutions.

Reinforcement Learning (RL) could have been applied to learn complete transport layer functions [157, 174, 214] and more specifically error control [202, 206]. RL models learn to take *actions* that maximize a *reward function* in an *environment* [207]. With such a general problem abstraction, RL can be applied to a broad range of problems. In the particular case of search algorithm design for adaptive HARQ, RL could be used to decide when to encode (i.e., block length), and when to transmit parity packets (i.e., repair schedule). However, the results here presented discourage such an approach. Sec. 5.4 has shown that learning the complete set of parameters (k, p, N_C) unnecessarily increases the neural network size and hence the inference time. Therefore, it stands to reason that learning yet another parameter, namely N_P, is likely to increase the network size even further.

The effect of increasing network size could be compensated with more advanced layers. The fully connected layer has been selected because no a priori structure was found in the data that could be exploited as other network architectures such as CNNs do. The memory mechanism implemented in LSTM networks makes it a good candidate, as it can remember previous actions in the learning phase better than a plain fully connected network, thereby improving its feature extraction capabilities. However, LSTMs also have a more complex structure [191] that could lead to longer training and low inference.

Whether a pure learning solution exists that would be able to achieve the predictably lower inference times is still an open question that is left for future work. In the meantime, DeepSHARQ has achieved unprecedented performance levels when it comes to providing a predictably reliable, real-time transport. By striking the right balance between learning and algorithmic components, DeepSHARQ has reduced the complexity of the learning problem to learning a regularized version of the block length while leaving the remaining parameters to optimized algorithmic implementations with low computational complexity.

5.6 Related Work

Given their good performance in other fields, it was just a matter of time before deep learning models spread into the networked communications field as well. At its core, the Internet protocol stack is designed to support a diversity of applications and communication technologies [137]. Two well-known drawbacks imposed by such an assumption are i) the presence of conflicting optimal operation points [83, 89, 169], and ii) the need to generalize to a wide range of network conditions [120, 126, 145, 184]. Adapting the protocols to particular use cases has been common practice when facing these two issues. The different versions of TCP for every generation of mobile communication [106, 119] are a clear example.

Thanks to their efficient feature extraction capabilities, DNNs have the potential to learn different policies that adapt to the underlying channel [126, 155, 214] and provide different quality levels depending on the application [202, 206, 210]. Despite their proven better performance, transport protocol updates usually take several years to be widely deployed [137, 148]. Instead of redesigning the system as new applications and communication channels develop, DL models can use *online learning* to retrain for a few epochs when new conditions are detected that significantly deviate from samples in the training dataset [174, 210]. In the highly diverse ecosystem the Internet has become in the last decades, DNNs provide richer adaptation policies with better adaptation to unseen conditions.

DL has already been applied to the parameterization of different error control schemes, although always assuming a pure FEC scheme [171, 202, 206]. DeepRS [171] implements a Reed-Solom FEC scheme powered by an LSTM network that predicts future loss patterns in a block. Although the neural network fails to predict the exact packets that will be lost, it achieves high accuracy in predicting the number of losses in the block, which the authors use as an estimator of the required amount of RI. In the context of real-time

video streaming, Chen et al. [202] select the amount of redundancy in FEC based on a video quality metric. The DNN trained by the authors maps video quality to RI, thereby achieving higher loss protection levels the more relevant the frames are. While the two aforementioned articles apply DL to block codes, Emara et al. [206] trained an LSTM network to infer the parameters of streaming codes.

The application of deep learning to congestion control has also been fruitful. To the best of our knowledge, [157] is the first deep-learned congestion control algorithm, which was trained on emulated network traces. Eagle [174] applies imitation learning for a faster training phase. Similar to how humans learn from teachers, Eagle uses BBR [112] to guide its training until a model is learned that outperforms BBR. Most similar to the approach presented in this chapter are, to the best of our knowledge, [210] due to its application interface to state the requirements and [214] due to its fast reaction to channel changes. Ma et al. [210] designed a congestion control algorithm that adapts its performance to specific application requirements—i.e., throughput, delay, and loss rate. As the requirements are included in the input to the model, the model can learn independent policies for different application types, thereby avoiding the aforementioned conflicting operating points issue. The proposed algorithm must generalize to new applications with different requirements that were not considered in the training. The authors leverage transfer learning to quickly transfer the knowledge acquired to new applications. In [214] the authors propose a two-step mechanism to congestion control in order to avoid the frequent computationally expensive execution of the DNN. Upon an ACK reception, a traditional congestion algorithm is executed to obtain a quick reaction to congestion signals. On a larger time scale, a policy adaptation DNN runs that fine-tunes the congestion algorithm to the channel conditions.

Channel estimation requires statistical algorithms that exploit temporal relations in the data to predict future channel states. LSTM networks excel precisely at this task thanks to their feedback mechanisms that introduce memory in the network. DeepQ [146] implements a deep generative network with LSTM for QoS estimation in datacenter networks. The feedback mechanism in LSTM networks makes them more computationally expensive than other DNNs—e.g., fully connected or convolutional networks. Instead of training a single, large LSTM model, LightFEC [191] predicts packet loss patterns with a divide and conquer approach: channel states producing similar loss patterns are clustered together, and a smaller network is trained for each of them. Finally, weight pruning is applied to further reduce the model size.

Deep learning has successfully been applied to other layers of the stack apart from the transport layer. Adaptive video streaming has been one of the first areas in which deep reinforcement learning was applied for networked systems. Pensieve [126] and Comyco [155] automatically learn ABR policies for DASH. Other areas worth mentioning are the optimization of wireless systems [168] or fast traffic analysis [213].

5.7 Conclusion

SHARQ's efficient solution space exploration and repair schedule construction enable it to find the optimal HARQ configuration significantly faster than previous search algorithms. However, its inference time is too high to provide predictable performance on resource-constrained embedded systems. The advent of DL presents a unique opportunity to bring adaptive HARQ to embedded systems thanks to the ability of DNNs to extract patterns from large datasets and automatically learn mathematical functions that minimize a target cost function. This chapter has introduced DeepSHARQ, a DL-based search algorithm that applies expert knowledge to target the specific parts in the search pipeline that would benefit from a DL approach the most. DeepSHARQ combines learning and algorithmic components with a novel output regularization mechanism to further reduce the learning complexity of the problem. As a result, DeepSHARQ achieves high accuracy with significantly smaller DNNs than if the complete problem would have been modeled as a learning problem. The evaluations presented in this chapter have shown that DeepSHARQ achieves unprecedented predictable performance on embedded systems with limited computational capabilities.

Chapter 6

Energy-Aware Adaptive HARQ

The cost and energy efficiency of embedded devices have made them the natural component of CPSs [54, 116]. Energy efficiency, in particular, is key to guarantee a sustainable deployment of CPSs and a large operation lifespan once deployed. CPSs are expected to permeate our society in what is typically known as the Internet of Things (IoT) [85]. In an age of climate crisis due to the effects of greenhouse gases, taking sustainability aspects into account in CPS design is of utmost importance [220, 216], given the CO_2 intensity of our current power grids. Moreover, as embedded devices are frequently battery-powered, an energy-aware operation extends their battery life, thereby enabling them to operate for a longer time. This chapter looks at PRRT, and more precisely its error control function, from this standpoint.

MDS codes are at the core of PRRT's error control (see Sec. 3.3), which perform operations in high-order fields—i.e., $GF(2^8)$ in the case of PRRT—with little hardware support on embedded devices. Although algorithmic optimizations have been proposed to alleviate this lack of support [26, 52], the results in this chapter show that PRRT experiences large en-/decoding delays when deployed on embedded devices. This new delay source must be accounted for in the delay budget. A case study is presented that systematically analyzes the introduction of binary codes as a more energy-efficient alternative to MDS codes for the deployment of PRRT on low-end devices.

6.1 The Complexity Dilemma (Revisited)

As discussed in more detail in Sec. 2.3.1, block codes perform a matrix-vector multiplication to encode a message $\mathbf{m}$ with k symbols into a codeword $\mathbf{c}$—i.e., $\mathbf{c} = \mathbf{m} \cdot \mathbf{G}$, where $\mathbf{G}$ is a $k \times n$ generator matrix. This operation has a computational complexity $\mathcal{O}(kn)$. At the receiving end, the original message is decoded by solving $\mathbf{m} = \hat{\mathbf{c}} \cdot \hat{\mathbf{G}}^{-1}$, where $\hat{\mathbf{c}}$ is a subset of k symbols of $\mathbf{c}$ and $\hat{\mathbf{G}}$ is a $k \times k$ submatrix of $\mathbf{G}$. The matrix inversion has complexity $\mathcal{O}(k^3)$ if the Gauss-Jordan algorithm is employed.[1] Motivated by the

[1] For matrices fulfilling certain characteristics, other algorithms bring the complexity of matrix inversion down to $\sim \mathcal{O}(2)$. Nevertheless, PRRT implements Gauss-Jordan and hence this algorithm is considered in the following

computationally expensive matrix inversion in MDS codes, which makes it difficult to achieve high data rates, codes have historically changed from perfect MDS codes towards imperfect binary codes that introduce a higher error floor but can be implemented more efficiently.

6.1.1 MDS Codes

The principal characteristic of MDS codes is that they fulfill the Singleton Bound with equality: $d_{min} = e + 1$ where d_{min} is the minimum distance between codewords and $e = n - k$ the number of correctable erasures [16].

Definition 6.1.1 (Hamming Distance)**.** The Hamming Distance $d_H(x, y)$ is defined as the number of different symbols between the codewords x and y.

Definition 6.1.2 (Minimum Distance)**.** The Minimum Distance d_{min} of a block code is defined as the minimum Hamming Distance between any two codewords x and y in $\mathcal{C}$.

$$d_{min} = min\{d_H(x, y) \mid x, y \in C, x \neq y\} \tag{6.1}$$

Fulfilling the Singleton Bound with equality guarantees that the number of correctable losses equals the number of coded parity packets. To be able to correct e erasures with any k received symbols, any $k \times k$ submatrix of $\mathbf{G}$ must be invertible. Out of the available mechanisms to construct such a generator matrix [22, 26], PRRT implements systematic Vandermonde MDS codes in $GF(2^8)$ (see Sec. 3.3.3 for more details). Therefore, the analysis in the following focuses on this type of code without loss of generality, since the basic operations remain the same regardless of the matrix construction.

6.1.2 Binary Codes

Unlike MDS codes, binary codes do not fulfill the Singleton Bound with equality, which is a direct consequence of being defined in the binary field $GF(2)$. Linearly independent columns in the generator matrix must be constructed by introducing the 0th element and thus parity packets do not carry information from all the input symbols in the message $\mathbf{m}$. Therefore, binary generator matrices do not ensure that any $k \times k$ submatrix is invertible. As a result, binary codes have a residual decoding error probability even when k or more packets are received, and require excess packets to approach the channel capacity and achieve the same reliability level as MDS codes. On the other hand, the computational complexity of binary codes is lower because i) the matrix-vector operation can be implemented with simple XORs instead of arithmetic operations in higher-order fields, ii) they need fewer operations on average to construct the parity packets due to the presence of the 0th element in $\mathbf{G}$, and iii) graph-based algorithms can be used for decoding that do not explicitly invert the generator matrix [16]. The *complexity dilemma* refers to this trade-off between redundancy excess and low complexity.

Table 6.1: Binary codes and their adoption in cellular networks.

Code	Cellular generation	Reference
Turbo	3G, 4G	[23]
LDPC	4G, 5G	[15]
Polar	5G	[60]

Luby showed that the portion of the RI stemming from the higher error floor becomes negligible for sufficiently large block lengths [32] and proposed a binary code construction that reduces the complexity of the decoding operation to $\mathcal{O}(n \cdot log(n))$. Given the trend towards increasing data rates with every new wireless generation, binary codes have become the standard for error coding in the physical layer—see Table 6.1.

6.1.3 The Role of the Transport Layer

In IP networks, bit flips in a packet prevent it from being forwarded to the upper layers of the protocol stack and entire packets may be dropped in congested routers. Therefore, although HARQ in the lower layers operates directly on bits, at the transport layer it should recover entire packets. PRRT addresses this issue with a virtual interleaver, which iterates the same coding operation throughout the complete packet length (see Secs. 2.3.1 and 3.3.3). As multiple symbols are grouped in each packet, and k packets must be collected before encoding, applications with tight delay constraints allow for a maximum block length that is several orders of magnitude smaller than in the physical layer. For example, a video conference stream generating data at 10 Mb/s with a target delay of 100 ms can collect 83 packets at maximum with the typical MTU in IP networks of 1,500 B. In contrast, typical block lengths in the lower layers are in the order of 10,000 symbols [32, 45]. While a single matrix inversion is required per block, the matrix-vector multiplication is iterated throughout the packet length. For the systematic Vandermonde code implemented in PRRT, Sec. 3.3.3 derives the computational complexity of the matrix-vector multiplication (Eq. (6.2)) and matrix inversion (Eq. (6.3)), where $e \leq min(k, n-k)$ is the number of lost packets and L_p the packet length.

$$\mathcal{C}_{mul}^{MDS} = \mathcal{O}(keL_p) \tag{6.2}$$

$$\mathcal{C}_{inv}^{MDS} = \mathcal{O}(ke^2) \tag{6.3}$$

The computational complexity of the encoding and decoding in an MDS can also be expressed as a function of the basic algebraic operations defined in $GF(2^m)$, namely multiplication ($\texttt{mul}_q$), division ($\texttt{div}_q$), subtraction ($\texttt{sub}_q$), and addition ($\texttt{add}_q$)—in $GF(2^m)$ the addition and subtraction can be obtained by a binary XOR and hence are equivalent. While both encoding and decoding perform a matrix-vector multiplication (Eq. (6.4)), decoding performs a matrix inversion as well (Eq. (6.5)). The Gauss-Jordan algorithm performs subtractions and divisions, and has the computational complexity in Eq. (6.6) [19].

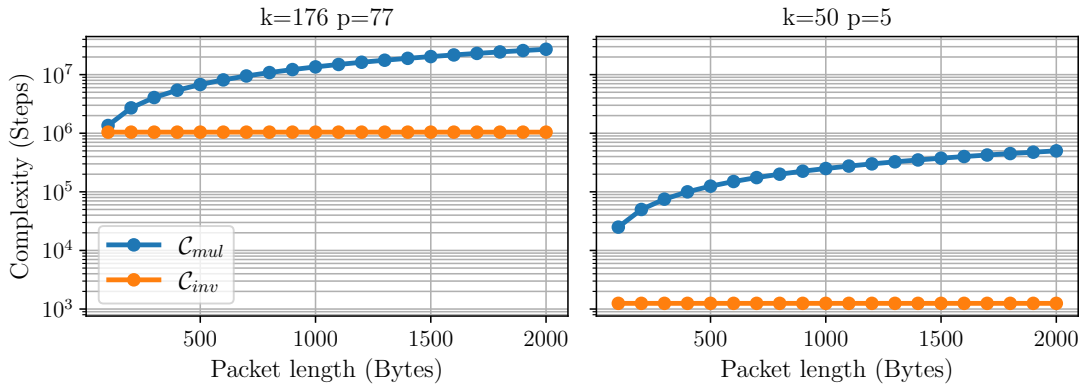


Figure 6.1: Computational complexities $\mathcal{C}_{mul}$ and $\mathcal{C}_{inv}$ as a function of the packet length parameterized by k and p.

$$\texttt{enc}_{mds}(k, p, L_p, m) = (k \cdot \texttt{mul}_q + (k-1) \cdot \texttt{add}_q) \cdot p \cdot \left\lceil \frac{L_p}{m} \right\rceil \tag{6.4}$$

$$\texttt{dec}_{mds}(k, p, L_p, m) = \texttt{enc}_{mds}(k, p, L_p, m) + \texttt{inv}_q(k) \tag{6.5}$$

$$\texttt{inv}_q(k) = k^2 \cdot \texttt{div}_q + \frac{k^3 - k}{2} \cdot (\texttt{mul}_q + \texttt{sub}_q) \tag{6.6}$$

Given an MDS code in $GF(2^8)$, $\left\lceil \frac{L_p}{m} \right\rceil = MTU$ in bytes since symbols are one-byte long. For a time-constrained protocol such as PRRT, $MTU \gg k$ and $MTU \gg p$. It directly follows that $\mathcal{C}_{mul} \gg \mathcal{C}_{inv}$ [7, 8]. Fig. 6.1 compares the complexity of the two matrix operations parameterized by k and p. On the one hand, $k = 176$ and $p = 77$ have been selected because it is the configuration in the dataset in Sec. 3.6.1 with the highest $\mathcal{C}_{inv}$. On the other hand, $k = 50$ and $p = 5$ have been selected because it is a more representative configuration of the dataset, as shown in Fig. 6.2, which depicts the probability distribution of the optimal block length and number of parity packets found by the SHARQ algorithm (see Sec. 4.3). Even for the highest complexity of the matrix inversion, the complexity of the multiplication is an order of magnitude higher for $MTU = 1,500$ B.

Multiple proposals bringing binary codes to the transport layer have been published in recent years [8, 138, 151]. However, none of them systematically analyzes how the transport layer has turned the tables when it comes to the complexity dilemma: the matrix inversion no longer dominates the computational complexity and the redundancy excess cannot be compensated with large block lengths. The remaining advantages of binary codes over MDS are the sparse generator matrix and the purely XOR-based parity packet creation. The remainder of this chapter explores whether these advantages still

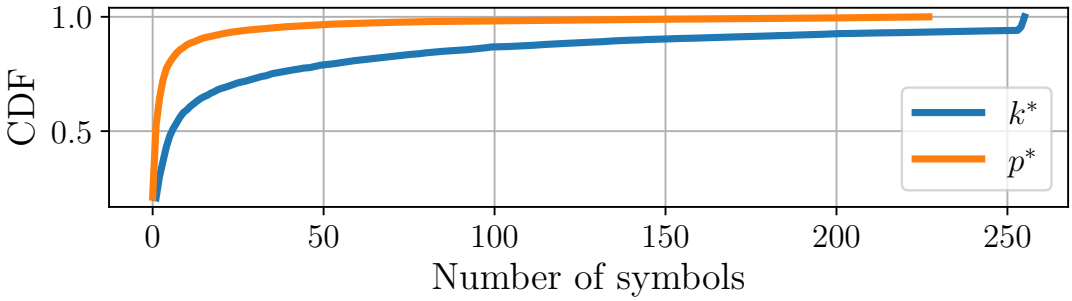


Figure 6.2: Cumulative Distribution Function (CDF) of the optimal block length and number of parity packets in the dataset in Sec. 3.6.1.

compensate for the redundancy excess, thereby resolving the complexity dilemma in favor of or against binary codes in the transport layer as well.

6.2 Binary Erasure Codes

Linear block codes operating in $GF(2)$ are called *binary codes* as symbols are one-bit long. Since the generator matrix $\mathbf{G}$ contains the zero element, the multiplication of the message vector $\mathbf{m}$ and the i-th column of the generator matrix ($i \in [0, p-1]$) boils down to the XOR of the input packets in those positions where the column contains a one. Formally,

$$c_i = \oplus\{m_x \mid 0 \leq x \leq k-1 \ \wedge \ G_{x,i} == 1\} \tag{6.7}$$

Consequently, the number of operations in the multiplication is proportional to the *degree* (d) of the column, i.e., the number of 1s in that column. Although the symbol length is a single bit, the XOR operation has the advantage that multiple symbols can be grouped in a single CPU instruction. Therefore, the number of performed XOR operations ($\texttt{xor}_b$) depends on the CPU architecture, where b is the bit width of the CPU. For example, $b = 32$ for a 32-bit architecture and $b = 64$ for a 64-bit architecture. The number of operations in the encoding function of a binary code is

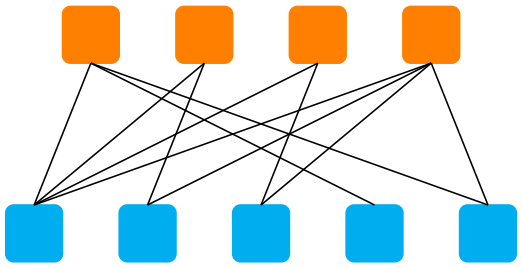
$$\texttt{enc}_{bin}(\bar{d}, p, L_p, b) = (\bar{d} - 1) \cdot \texttt{xor}_b \cdot p \cdot \left\lceil \frac{L_p}{b} \right\rceil \tag{6.8}$$

where $\bar{d}$ is the average column degree of the generator matrix, which depends on the binary code construction (more on this later).

The binary generator matrix can be represented as a bipartite *Tanner graph* [95], as depicted in Fig. 6.3. For input symbol decoding, the *Message Passing* (MP) algorithm [40] is executed on this graph, which performs back-substitution without the need for an explicit matrix inversion. The algorithm begins with a received symbol of degree one, which already contains a verbatim copy of one of the input symbols in the message vector $\mathbf{m}$. The decoded input vector is then propagated into the graph by XORing it

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

(a) Matrix representation of a binary generator matrix



(b) Tanner graph representation of a binary generator matrix

Figure 6.3: Binary generator matrix (a) and its representation as a Tanner graph (b).

to the received symbols it is connected with, thereby reducing their degree by one. This process is iterated until either all input symbols have been decoded or no degree-one received symbol is found. Similarly to the matrix inversion for MDS codes, the overhead to find the degree-one columns is negligible for $L_p = MTU = 1,500$ B (see Sec. 6.4), and thus the decoding complexity is upper bound by the encoding complexity:

$$\texttt{dec}_{bin}(\bar{d}, p, L_p, b) \leq \texttt{enc}_{bin}(\bar{d}, p, L_p, b) \tag{6.9}$$

Not all excess packets are necessarily used in decoding, hence the inequality between the two complexity terms.

Unlike MDS codes, binary codes do not have a canonical mechanism for building the generator matrix. Randomly generated binary codes [28, 32, 40, 45] set the entries of the generator matrix to one or zero following a probability distribution for the column degree. This degree distribution is designed such that the probability of the randomly generated matrix not being invertible (δ) is below a target threshold. Random fountain codes [40] apply a uniform degree distribution, thereby achieving an average degree $\bar{d} = \frac{k}{2}$. Luby Transform (LT) codes [32] use an ideal degree distribution that reduces the density of $\mathbf{G}$ and the number of excess packets. Raptor codes [45] further reduce the density of the generator matrix. In order to recover from the higher error floor that a low-density matrix introduces, raptor codes implement a pre-coding step with an optimal block erasure code.

6.2.1 Random Fountain Codes

The metaphor of the fountain refers to an encoder that generates an unlimited number of coded symbols (i.e., drops) [28] $n \to \infty$, where the coded symbols are generated by XORing a randomly chosen number of input symbols. Since the encoder can indefinitely keep generating coded symbols, this kind of code is known as *rateless* ($r = \frac{k}{n} \to 0$). As mentioned before, fountain codes reduce the complexity when compared to MDS codes due to their sparse binary matrix and the absence of an explicit matrix inversion. The receiver can obtain the k input symbols with a subset $k' \geq k$ of the coded symbols,

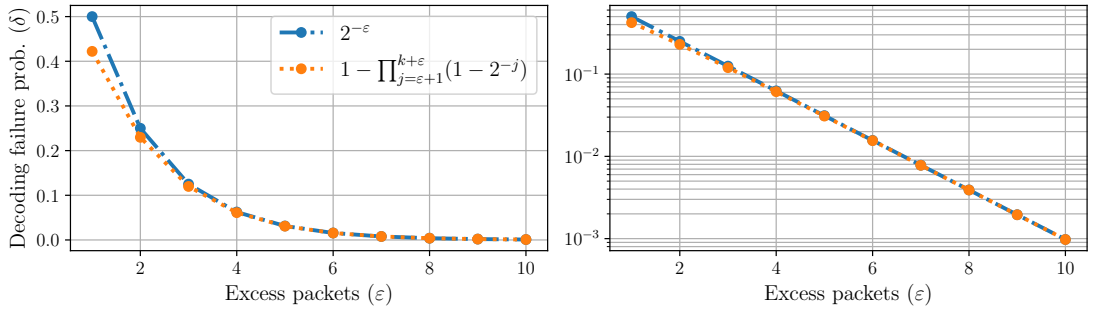


Figure 6.4: Probability of decoding failure of a random fountain code as a function of the number of excess packets for $k = 10$.

for which it requires information on how the parity packets were encoded in the first place—i.e., how the generator matrix was constructed. This information can either be exchanged with some signaling mechanism with low overhead or using pseudo-random codes with a common seed for the sender and receiver.

In *random fountain codes* as described in [40], input symbols are included in a parity packet with 50% probability. Therefore, it has an average column degree $\bar{d} = \frac{k}{2}$. MacKay [40] proves that the probability of a random $k \times n$ binary matrix to contain an invertible $k \times k$ matrix is bounded by

$$1 - \prod_{j=\varepsilon+1}^{k+\varepsilon} (1 - 2^{-j}) \leq \delta \leq 2^{-\varepsilon} \tag{6.10}$$

where ε is the number of excess packets such that $n = k + \varepsilon$. Fig. 6.4 shows that both bounds converge very quickly even for a small block length of $k = 10$. Therefore, the required number of excess packets for random fountain codes can be expressed as

$$\varepsilon_{rand} = log_2\left(\frac{1}{\delta}\right) \tag{6.11}$$

Input packet losses no longer occur only when less than k packets are received, but even when the receiver gets k or more packets, but it cannot invert any $k \times k$ submatrix. This new source of unreliability must be considered when implementing a predictably reliable transport service based on binary codes. Eq. (6.13) provides the probability of losing i input packets with a binary code, which differs in two aspects from the same probability for an MDS code (see Eq. (3.18)): i) the lower bound for the number of erasures e, and ii) the term $\delta(n, k, e)$. Packets can be lost even with $e \leq k$ if the matrix cannot be inverted, and $\delta(n, k, e)$ models the probability that the receiver has no invertible $k \times k$ after e packets losses (see Eq. (6.14)). If more than p packets are lost, $\delta(n, k, e) = 1$ because the receiver cannot construct a square matrix. If p or fewer packets are lost, the probability of decoding failure depends on the number of received packets beyond k, as shown in Eq. (6.10). The latter case is modeled with the upper bound in Eq. (6.10), which results in a more conservative parity packet selection.

$$PLR_{HARQ}^{bin}(k,p) = \frac{1}{k}\sum_{i=1}^{k} i \cdot Pr(I_k = i)|_{bin} \tag{6.12}$$

$$Pr(I_k = i)|_{bin} = \sum_{e=i}^{n-k+i} \delta(n,k,e) \cdot \binom{n}{e} {p_e}^e (1-p_e)^{n-e} \cdot p_d \binom{e}{i} \tag{6.13}$$

$$\delta(n,k,e) = \begin{cases} 2^{-(n-k-e)} & \text{if } 0 \le e \le n-k \\ 1 & \text{if } n-k < e \le n \end{cases} \tag{6.14}$$

$$p_f^{bin}[c] = \sum_{i=max(0,\mathbf{n}[c-1]-p)}^{k-1} \delta\big(\mathbf{n}[c-1], k, \mathbf{n}[c-1]-i\big) \cdot \binom{\mathbf{n}[c-1]}{i} (1-p_e)^i {p_e}^{\mathbf{n}[c-1]-i} \tag{6.15}$$

For random binary codes to be fully integrated into the SHARQ algorithm in Sec. 4.3, the probability of a cycle to be triggered needs to be adapted to consider the decoding failure probability as well (see Eq. (6.15), and Eq. (3.13) for the MDS counterpart). The term $\delta\big(\mathbf{n}[c-1], k, \mathbf{n}[c-1]-i\big)$ models the decoding failure probability for the $k \times \mathbf{n}[c-1]$ generator matrix that can be built for all the packets sent up to the $(c-1)$th cycle. It can be proved that $\delta\big(\mathbf{n}[c-1], k, \mathbf{n}[c-1]-i\big) = 1 \; \forall \; i \in [max(0, \mathbf{n}[c-1]-p), k-1]$, and hence $p_f^{bin}[c] = p_f[c]$.

6.2.2 Luby Transform Codes

Luby Transform (LT) codes [32] also belong to the family of random binary codes, but they implement a more sophisticated degree distribution than random fountain codes. The design of this kind of code is motivated by the *LT process* described in [32]: Initially, all input symbols are uncovered. The decoding process with the MP algorithm described above must always start with a degree-one node, which in turn covers an input symbol. The set of all degree-one nodes is called the *ripple*, which contains all covered input symbols that have not been processed yet. In every step, a node is extracted from the ripple and processed by XORing it with all neighboring nodes—i.e., all nodes it shares an edge with. This process may uncover more degree one-nodes, thereby increasing the ripple size. The LT process ends when the ripple is empty. The objective of LT codes is designing a degree distribution $\rho(d) \; \forall \; k \in [1,d]$ that provides a high success probability by maintaining the ripple full until all input symbols are covered. The design objectives for the degree distribution described in [32] are a clear example of the complexity dilemma introduced in Sec. 6.1: the degree distribution should produce i) as few encoding symbols as possible to achieve a high decoding probability (i.e., minimize the required excess packets), and ii) an average degree of the generator matrix that is as low as possible (i.e., minimize the computational complexity).

Luby proposed two different degree distributions. The *Ideal Soliton distribution* (see Eq. (6.16)) displays ideal behavior in terms of the excess parity packets required to

decode all input symbols. However, this distribution is very fragile because it adds input symbols to the ripple at the same time as they are processed. Maintaining the ripple size of one makes this ideal distribution very sensitive to variances from the expected behavior.

$$\rho(d) = \begin{cases} \frac{1}{k} & \text{if } d = 1 \\ \frac{1}{d(d-1)} & \text{if } 2 \leq d \leq k \end{cases} \tag{6.16}$$

On the other hand, the *Robust Soliton distribution* (see Eq. (6.19)) ensures that a large size of the ripple is maintained throughout the whole process with high probability. Instead of 1, the expected number of degree-one nodes is S defined in Eq. (6.17), where δ is the target decoding failure probability and $c > 0$ a free parameter ($c < 1$ is known to achieve good performance [40]).

$$S = \left\lceil c \cdot ln\left(\frac{k}{\delta}\right)\sqrt{k} \right\rceil \tag{6.17}$$

$$\tau(d) = \begin{cases} \frac{S}{k}\frac{1}{d} & \text{if } 1 \leq d \leq \frac{k}{S} - 1 \\ \frac{S}{k} log\left(\frac{S}{\delta}\right) & \text{if } d = \frac{k}{S} \\ 0 & \text{if } d > \frac{k}{S} \end{cases} \tag{6.18}$$

$$\mu(d) = \frac{\rho(d) + \tau(d)}{Z} \tag{6.19}$$

$$\varepsilon_{LT} = \mathcal{O}\left(2 \cdot ln\left(\frac{S}{\delta}\right) S\right) \tag{6.20}$$

With the Robust Soliton distribution, the LT codes have an average degree distribution $\bar{d} = \mathcal{O}\left(ln\left(\frac{k}{\delta}\right)\right)$ and the required excess packets to achieve a probability of decoding failure δ is shown in Eq. (6.20) [40].

6.2.3 Multicast-Enabled Polar Codes

With the advent of swarm robotics [128, 159, 185], timely information needs to be distributed to large groups of receivers [226]. The computationally efficient binary codes are ideal in this scenario, in which the robots in the swarm typically have low computational power. Unlike MDS codes, parity packets in binary codes do not include information from every input symbol because the zero element is present in the generator matrix. In unicast scenarios, the decoding failure probability δ can be made arbitrarily low by including excess packets. However, reducing the column degree is undesirable in multicast scenarios where the probability of every packet being lost by at least one receiver is high. When multiple receivers experience different loss patterns, requesting a new randomly generated parity packet does not ensure that the information they are missing will be transmitted. Assume n packets are transmitted to M receivers, the probability that every packet is lost by at least one of the receivers is shown in Eq. (6.21). For simplicity,

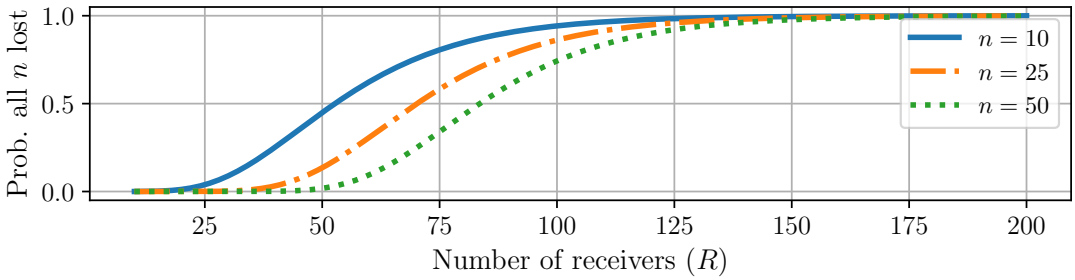


Figure 6.5: Probability that all n packets in the block are lost by at least one receiver in multicast on an i.i.d. channel with $p_e = 0.05$ as a function of the number of receivers M.

Eq. (6.21) assumes a BEC with the same packet erasure rate for every receiver in the multicast group. Fig. 6.5 depicts Eq. (6.21) for different values of n in the short block length regime, with all probabilities approaching 1 in the low one hundred range.

$$Pr(\text{All } n \text{ lost multicast}) = \left(1 - (1 - p_e)^M\right)^n \tag{6.21}$$

Included in the 5G standard, polar codes [60] implement a deterministic matrix construction that can be exploited to improve the performance of binary codes in multicast [8]. Polar codes exploit the polarization effect of binary codes: a binary code can be seen as a combination of n independent binary channels whose capacity polarizes to either almost perfect or almost fully noisy. If the columns of the $n \times n$ polar matrix correspond to the different channels, then the row in the matrix with the most 1s disperses the source symbol over the most channels, thereby achieving the highest channel capacity. Given the polarization effect, polar codes complement the input symbols with frozen symbols located in the positions with the lowest channel capacity. Frozen symbols are typically fixed to 0, and since their position is known, at the receiver end they convey information about the channel and can be used to enhance the decoding process. However, when polar codes are used in erasure channels, frozen symbols only convey information when they are erased and thus their transmission can be omitted as they do not enhance the decoding capabilities.

Arikan [60] proposed the *Bhattacharyya parameter* to measure the quality of the different channels in the generator matrix. Given $W : \mathcal{X} \to \mathcal{Y}$ a generic binary-input discrete memoryless channel with input alphabet $\mathcal{X}$, output alphabet $\mathcal{Y}$, and transition probabilities $W(y|x), x \in \mathcal{X}, y \in \mathcal{Y}$, the Bhattacharyya parameter is defined as

$$Z(W) \triangleq \sum_{y \in \mathcal{Y}} \sqrt{W(y|0)W(y|1)} \tag{6.22}$$

An $n \times n$ polar matrix operates on n independent W channels to create a set of polarizing channels $\{W_n^{(j)} : 1 \leq i \leq n\}$, where $n = 2^i$ and $i \geq 0$. The smallest combination of channels is W_2, which is generated with the matrix

$$\boldsymbol{F} \triangleq \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{6.23}$$

More generally, the polar matrix $\mathbf{G}_n = \boldsymbol{B}_n \boldsymbol{F}_n^{\otimes j}$, where $\boldsymbol{B}_n$ is the bit-reversal permutation matrix and $\boldsymbol{F}_n^{\otimes j}$ is the Kronecker power of the matrix $\boldsymbol{F}$. The Kronecker power $\boldsymbol{A}^{\otimes j}$ is defined as $\boldsymbol{A} \otimes \boldsymbol{A}^{\otimes(j-1)}$ for all $j \geq 1$, where

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} A_{11}\boldsymbol{B} & \cdots & A_{1j}\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ A_{i1}\boldsymbol{B} & \cdots & A_{ij}\boldsymbol{B} \end{bmatrix} \tag{6.24}$$

is an $i \times j$ matrix resulting from the Kronecker product of the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$. In the particular case of the BEC, this polar matrix construction allows for a recursive calculation of the Bhattacharyya parameter:

$$Z\left(W_n^{(2j-1)}\right) = 2Z\left(W_{n/2}^{(j)}\right) - Z\left(W_{n/2}^{(j)}\right)^2 \tag{6.25}$$

$$Z\left(W_n^{(2j)}\right) = Z\left(W_{n/2}^{(j)}\right)^2 \tag{6.26}$$

with $Z\left(W_1^{(1)}\right) = p_e$ and $1 \leq j \leq \frac{n}{2}$. [60] shows that a similar recursive approach can be used for en-/decoding with a computational complexity $\mathcal{O}(n \cdot log(n))$.

The matrix resulting from Kronecker power has a persymmetric structure—i.e., the matrix is symmetric with respect to the upper-right to the lower-left diagonal:

$$\boldsymbol{F}^{\otimes 4} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{6.27}$$

The last row in $\boldsymbol{F}^{\otimes 4}$ corresponds to the largest temporal dispersion, as it disperses the information over all the channels. Conversely, the first column in $\boldsymbol{F}^{\otimes 4}$ is the column with the largest dispersion over receivers in a multicast group, as it includes information from every input symbol that may be lost in the block. Therefore, the same Bhattacharyya parameter used to calculate each channel's reliability can be mirrored to measure the quality of each column for multicast. This property can be used to generate incremental redundancy for multicast groups.

Assume a vector $\mathcal{A}$ containing the indices of the k best channels in decreasing quality order as measured by the Bhattacharyya parameter, and the complementary vector $\mathcal{A}^C$ with the remaining $n - k$ channels in increasing quality order. For example, for the 16×16 polar matrix, the indices of the channel in decreasing quality order are

$$[16, 15, 14, 12, 8, 13, 11, 10, 7, 6, 4, 9, 5, 3, 2, 1]$$

so that $\mathcal{A} = [16, 15, 14, 12, 8, 13, 11, 10]$, and $\mathcal{A}^C = [1, 2, 3, 5, 9, 4, 6, 7]$. The multicast-enabled polar code creates a systematic generator matrix as follows: the worst channels are used for the frozen bits—i.e., the rows in $\mathcal{A}^C$ are ignored because they only contribute 0s to the packet XOR. Instead of using the channels in $\mathcal{A}$ as a non-systematic polar code would do, a precode is used that turns these channels into the systematic part of the matrix. Therefore, the k columns with indices in $\mathcal{A}$ are substituted by the columns in the $k \times k$ identity matrix. Since $\mathbf{G}^{-1} = \mathbf{G}$ in polar codes [60], this precode is equivalent to multiplying with the matrix $\mathbf{G}(\mathcal{A}, \mathcal{A})$. The parity packets are iteratively generated with the columns in $\mathcal{A}^C$. The persymmetric structure of the generator matrix means that ordering the channels in decreasing quality is equivalent to ordering the columns in increasing generalizability. Therefore, the parity packets should be transmitted in the order specified in $\mathcal{A}^C$, so that every parity packet transmission maximizes the number of receivers able to correct their losses.

To the best of our knowledge, there is no known analytical expression to obtain the probability of losing i data packets—i.e., $Pr(I_k = i)$—for polar codes. Therefore, for the evaluations presented in Sec. 6.3, this probability has been empirically calculated by constructing the polar matrix, emulating M receivers ([8] shows $M = 50,000$ achieves a low error estimation when compared to the ground truth), and calculating how many of them could recover from losses by running Gaussian elimination.

6.3 Theoretical Analysis

Random fountain, LT, and polar codes are by no means the only binary codes proposed in the literature. Still, they do represent the different trends that binary code design has experienced in the last decades. Random binary codes have gone from high average column degree (random fountain) to sparse generator matrix (LT), whereas the more recent polar codes deviate from sparsity with their deterministic matrix construction. This section presents a theoretical analysis of the three codes above to address the research question posed in this chapter, namely whether binary codes are a more energy-efficient alternative to MDS codes for time-sensitive transport protocols despite their RI overhead.

6.3.1 Complexity and Suboptimality

The packet loss probability model in Sec. 6.2.1 enables their computational complexity comparison with MDS codes. The computational complexity and the optimal number of parity packets for both codes are depicted in Fig. 6.6. The codes are compared in their systematic form and the optimal numbers of parity packets have been obtained as follows:

$$p_{opt}^{mds}(k) = min\{\, p \mid PLR_{HARQ}(k, p) \leq PLR_T\} \tag{6.28}$$

$$p_{opt}^{bin}(k) = min\{\, p \mid PLR_{HARQ}^{bin}(k, p) \leq PLR_T\} \tag{6.29}$$

Since the number of excess packets in random fountain codes is independent of k (see Eq. (6.11)), the gap between $p_{opt}^{mds}(k)$ and $p_{opt}^{bin}(k)$ becomes smaller as the block length

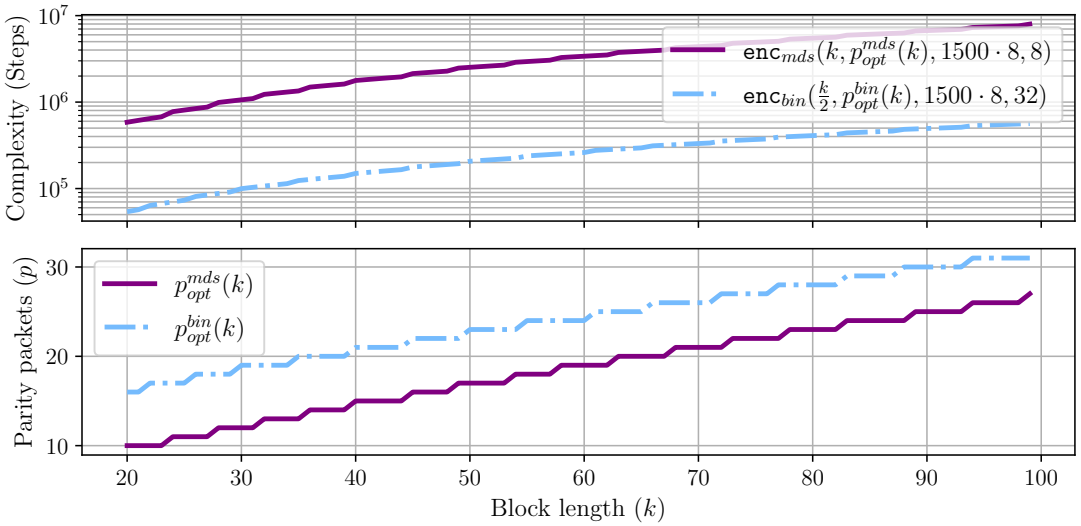


Figure 6.6: Comparison of the encoding complexity for MDS and random fountain codes for $PLR_T = 0.002$ and $p_e = 0.1$.

increases, as it can be observed in the second plot in Fig. 6.6. This property supports the aforementioned intuition that the relative portion to the RI used by the excess symbols diminishes for large block lengths.

Despite their parity packet overhead, the computational complexity of random fountain codes is an order of magnitude smaller for the complete range of block lengths depicted in the figure. This result suggests that binary codes could be an efficient alternative to MDS codes, even in the short block length regime of delay-bound transport protocols.

6.3.2 The Impact of Sparsity

Generator matrix sparsity is at the core of the complexity dilemma (see Sec. 6.1), as it affects the required number of excess packets and the number of edges in the Tanner graph, and hence the required XORs for en-/decoding. Fig. 6.7 compares random fountain and LT codes in terms of these two variables. The codes are compared in their systematic (i.e., only ε packets are encoded) and non-systematic (i.e., $k + \varepsilon$ packets are encoded) forms. The number of excess packets in LT codes has a log-linear relation with the free parameter c (see Eq. (6.20)). Providing a detailed analysis of this parameter is out of the scope of this thesis and hence it has been set to the frequently used value $c = 0.2$ [40].

Sparser binary generator matrices do not necessarily reduce the complexity of the code in the short block length regime. For small k values, LT's generator matrix has more edges than random fountain's in both forms, systematic and non-systematic, which is a result of the higher number of excess packets. As the block length increases, the

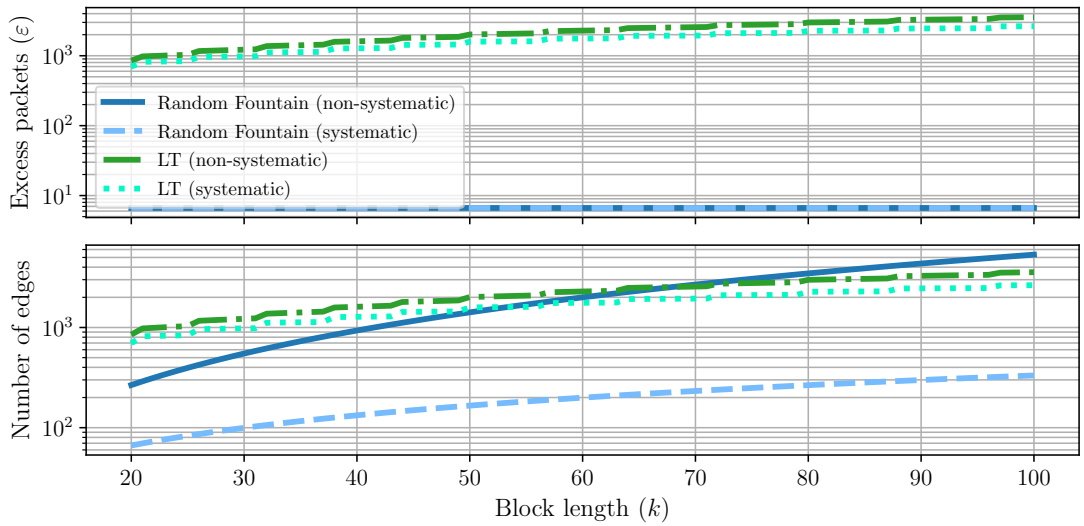


Figure 6.7: Number of excess packets that random fountain and LT codes require to achieve a decoding failure probability $\delta = 0.01$. The free parameter $c = 0.2$ has been configured for the LT code.

number of edges[2] in non-systematic random fountain codes grows quadratically with k due to their average column degree $\bar{d} = \frac{k}{2}$. This result is in line with the dominance of sparse codes in the lower layers of the protocol stack. However, random fountain codes in their systematic form outperform the other three configurations in terms of excess packets and number of edges. In the design of binary codes for the transport layer, non-sparse binary matrices producing a low number of excess packets are preferable to sparse codes.

6.3.3 Multicast Performance

To analyze the performance in multicast, a multicast group has been simulated with many receivers as different erasure patterns exist for $i \in [1, e]$ erasures in a block, where e is the number of correctable erasures in an MDS code transmitting $e = p$ parity packets. MDS codes are used as a reference as every parity packet can correct one erasure in every receiver, regardless of the pattern. After the transmission of every packet in the block, every receiver in the group performs Gaussian elimination to check how many erasures it can correct. Since each erasure pattern has a different probability, each receiver is weighted differently depending on the number of erasures it originally experienced. The probability of that erasure pattern with an erasure count of i occurring is

$$Pr(\text{pattern with } i \text{ erasures}) = {p_e}^i (1 - p_e)^{n-i} \tag{6.30}$$

[2]The number of edges for random fountain codes is $\frac{k}{2} \cdot (k + \varepsilon_{rand})$ in non-systematic form and $\frac{k}{2} \cdot \varepsilon_{rand}$ in systematic form. The edges in the graph for LT codes is $(k + \varepsilon_{LT}(\delta, k, c)) \cdot log(\frac{k}{\delta})$ in non-systematic form and $\varepsilon_{LT}(\delta, k, c) \cdot log(\frac{k}{\delta})$ in systematic form.

The weight for a receiver with i erasures is obtained with Eq. (6.31), in which the probability of that pattern to occur is normalized to the total probability of the patterns considered for the evaluation to occur (see Eq. (6.32)).

$$w_i = \frac{1}{w_T} Pr(\text{pattern with } i \text{ erasures}) \tag{6.31}$$

$$w_T = \sum_{i=1}^{e} \binom{n}{i} Pr(\text{pattern with } i \text{ erasures}) \tag{6.32}$$

The evaluation of random fountain codes must also account for the randomness in the generator matrix. An erased symbol is included in a parity packet with probability $Pr(symbol) = 0.5$, and hence the symbol is included in at least one of the p transmitted parity packets with probability $1 - Pr(symbol)^p$. A receiver can recover all its losses with a random fountain code $\mathcal{C}(n,k)$ if i) all e lost symbols are included within the transmitted p parity packets, and ii) the binary generator matrix is invertible. The probability of these two cases to occur is

$$Pr(\text{repair } e \text{ fountain}) = \big(1 - \delta(n,k,e)\big)\big(1 - Pr(symbol)^p\big)^e \tag{6.33}$$

and the receiver weight for random fountain codes is

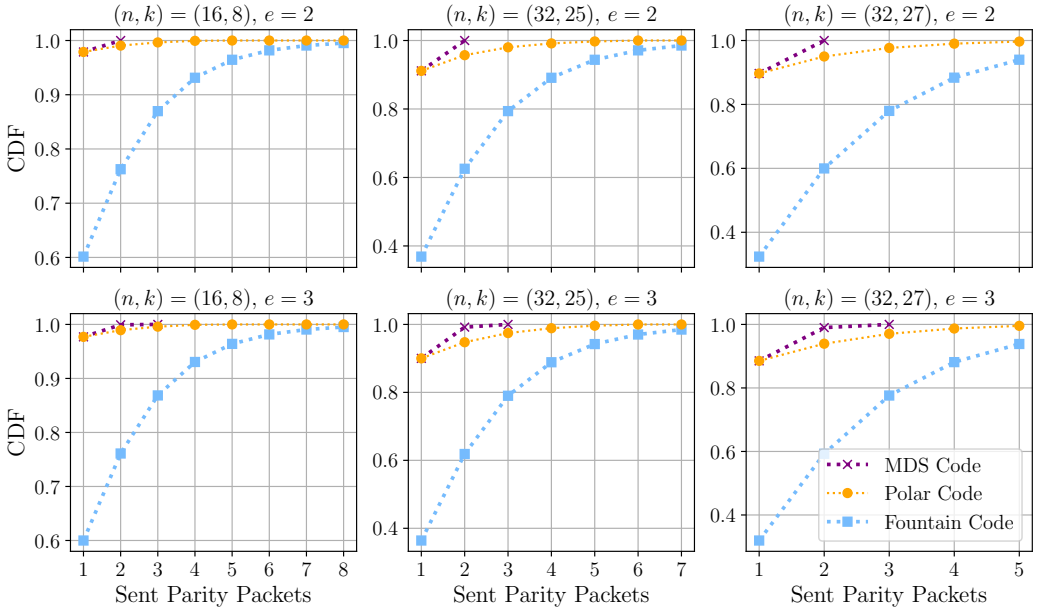
$$w_i^f = w_i \cdot Pr(\text{repair } e \text{ fountain}) \tag{6.34}$$

Fig. 6.8 shows the CDF of the erasure recovering probability for all receivers in the multicast group experiencing $i \in [1, e]$ erasures. The performance of the multicast-enabled polar codes proposed in Sec. 6.2.3 approaches the optimal performance of MDS codes. Although both binary codes converge as the number of parity packets increases, the first parity packets in random fountain codes provide a significantly lower receiver coverage on average due to their randomness.

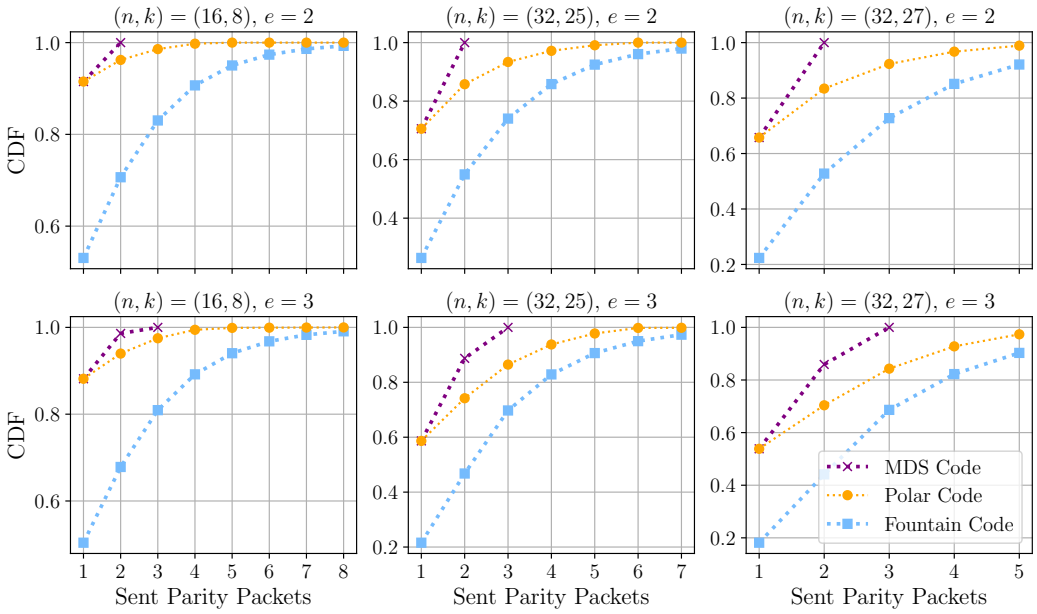
A closer look at the multicast-enabled polar codes shows that there is still room for improvement. Ordering the columns according to the Bhattacharyya parameter results in a decreasing column degree. For example, for $k = 10$ the column degrees are $[16, 8, 8, 8, 8, 4]$. As the frozen positions do not contribute any information to the parity packets, the column degree is reduced in practice to $[10, 6, 6, 7, 7, 3]$. However, since the degree difference is small, so is the expected gain of re-sorting. Therefore, this feature has not been implemented to avoid a code-rate-dependent reordering that would increase the complexity of the matrix construction.

6.3.4 Error Correction Capabilities

The final aspect to consider in the code comparison is their error correction capabilities. A WiFi-like simulation has been considered, with an erasure probability $p_e = 0.05$ as well as two different reliability targets—i.e., $PLR_T = \{0.01, 0.001\}$. Fig. 6.9 depicts the optimal number of parity packets—i.e., the minimum p that meets the PLR_T constraint—for MDS, multicast-enabled polar, and random fountain codes. Although



(a) $p_e = 0.01$



(b) $p_e = 0.05$

Figure 6.8: Cumulative Distribution Function (CDF) of the erasure recovering probability for all receivers in a multicast group.

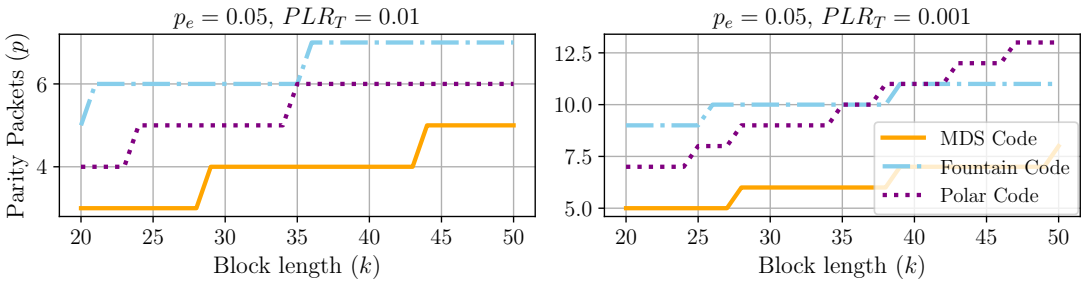


Figure 6.9: Required number of parity packets by MDS, multicast-enabled polar, and random fountain codes to meet a target packet loss rate $PLR_T = \{0.01, 0.001\}$ in a BEC with erasure probability $p_e = 0.05$.

the proposed polar code construction in Sec. 6.2.3 outperforms random fountain codes in the short block length regime, they require the most parity packets for large block lengths and a tight reliability constraint.

A drawback of the analysis here presented is the considered target packet loss rates—i.e., $PLR_T = \{0.01, 0.001\}$—, which are in the higher range of the application requirements the PRRT protocol is designed to support (see Table 3.2). The selection of these target values has been motivated by the employed methodology: due to the lack of a closed-form expression for the PLR for polar codes, Gaussian elimination must be evaluated for all possible combinations of lost packets in n. Therefore, this methodology becomes computationally intractable for the most stringent PLR_T values. Nevertheless, polar codes still have a lower parity packet overhead than random fountain codes in the short-block-length regime, which makes polar codes an interesting alternative for multicast applications in the IoT domain if the target delay is in the single-millisecond range. Therefore, the suitability of polar codes for packetized layers should be further analyzed in future work.

6.3.5 Conclusion

This section has shown that, despite their RI excess, binary codes are an interesting alternative to the optimal MDS codes due to their low complexity. Even in the transport layer operating in the short block length regime, where the redundancy excess accounts for a non-negligible portion of the overall data rate, random fountain codes reduce the computational complexity by an order of magnitude in comparison to MDS (see Sec. 6.3.1).

When addressing the complexity dilemma, the results presented in this section suggest there is not a one-size-fits-all solution that achieves the best performance—neither in terms of transmitted RI nor computational complexity—in every scenario. Sparse binary codes have been discarded due to their poor performance in the short block length regime (Sec. 6.3.2). However, the decision is not clear between multicast-enabled polar and random fountain codes. Polar codes approach the performance of MDS in multicast

(Sec. 6.3.3) and, although they require less excess packet with small block lengths than random fountain codes, they experience an excess packet explosion with applications demanding high reliability (Sec. 6.3.4). On the other hand, random fountain codes need more parity packets to achieve a high receiver coverage in multicast, but the required number of excess packets is more stable.

The analyzed codes achieve different trade-offs between network complexity (optimal redundancy vs. excess packets) and processing complexity (operations in high order fields vs. simple XOR operations). While optimizing for the transmitted RI solely focuses on the network complexity, the energy demand of the system seems a better alternative to find the code with the lowest resource footprint, regardless of where the complexity is put. However, no code seems to demand the lowest energy in all scenarios, which makes energy-aware error control a challenging optimization problem to solve Sec. 3.5.1. Not only should different codes be considered due to their heterogeneous performance depending on the application requirements and network conditions, but the device-specific energy demand should be considered. Precise energy models for a plethora of CPUs and network interfaces—including different wireless technologies, such as WiFi, Bluetooth, Bluetooth Low Energy, etc.—are required to make energy-aware decisions. The remainder of this chapter explores the idea of an energy-aware, adaptive HARQ implementation. For simplicity, the analysis has been limited to random fountain codes and a Raspberry Pi Zero W as the target code and platform, respectively. Extending the same analysis to other codes and platforms is left for future work.

6.4 Energy-Aware HARQ

Sec. 6.3.5 puts into question the suitability of the RI as the optimization metric for binary codes, which are suboptimal from the RI standpoint but reduce the processing time due to their lower computational complexity. The energy demand of the system seems a better optimization metric a priori, since it offers a single metric to account for the resource demand regardless of whether they are spent on the communication or processing part. Formally, given a code $\mathcal{C}(n,k)$ with $p = n - k$, the total energy demand that the code adds to the transmission of the k input symbols is

$$E(k,p) \leq \underbrace{E_{Tx}(k+p) + E_{Rx}(k+p)}_{\text{Communication resources}} + \underbrace{E_{encode}(k,p) + E_{decode}(k,p)}_{\text{Processing resources}} \tag{6.35}$$

where the upper bound stems from the fact that not all n packets are necessarily received due to erasures that may occur in the channel. The terms $E_{Tx}(p)$, $E_{Rx}(p)$, $E_{encode}(k,p)$, and $E_{decode}(k,p)$ correspond to the energy demand of transmitting, receiving, encoding, and decoding $k+p$ packets, respectively. Sec. 6.4.1 presents how these models have been built for the Raspberry Pi Zero W.

$$E_{HARQ}(k, N_C, N_P) = \frac{1}{k} \cdot E(k, N_P[0]) + \frac{1}{k} \sum_{c=1}^{N_C} p_f^R[c] \cdot E(k, N_P[c]) \tag{6.36}$$

Eq. (6.35) calculates the total energy demand when the packets are always encoded and transmitted. However, PRRT implements an incremental redundancy scheme, in which parity packets in the reactive cycles are only encoded and transmitted if triggered by the receiver. The energy demand of such an HARQ scheme is provided in Eq. (6.36) which is the energy counterpart of Eq. (3.11). The normalization by the block length allows the comparison between codes with different block lengths, as Eq. (6.36) de facto is the average energy demand increase per input symbol the HARQ configuration (k, N_P) introduces.

6.4.1 Energy Models

For the analysis of the energy efficiency of different coding techniques, energy models for the basic operations of a code must be constructed. The four basic operations are i) encoding, ii) decoding, iii) transmission, and iv) reception. The remainder of this section introduces the methodology followed to obtain energy measurements for these functions and the resulting energy models.

Methodology

The four aforementioned basic operations have been isolated from other protocol functions and executed on a Device Under Test (DUT), in this case, a Raspberry Pi Zero W. The Raspberry Pi Zero W runs the Raspbian Buster operating system with Linux Kernel 4.19, i.e. the same device employed in Chapter 4 and Chapter 5. The Raspbian operating system has been configured to run as few processes as possible, thereby reducing the noise experienced by the measurements. Executing the isolated basic coding operations allows for precise energy measurements.

The setup presented in [179] has been used to obtain the energy measurements. Fig. 6.10 presents a schematic of such a setup. The host PC sends execution commands to the DUT via WiFi. The energy measurement board intercepts the power supply at the USB of the DUT. The power measurements are obtained with a shunt resistor and the LTC2991[3] voltage and current monitor sensor, which has a 14-bit ADC. An Arduino Nano[4] microcontroller obtains the power draw from an Inter-Integrated Circuit (I2C) interface to the LTC2991 chip. The aggregated power measurements are communicated to the host PC via USB.

The host PC can send three different execution commands, each corresponding to an isolated piece of code. The *communication* command uses `iperf`[5] to transmit and receive data on a UDP socket. The *MDS code* command executes PRRT's MDS code implementation, which is described in detail in Sec. 3.3.3. Finally, the *random fountain code* command executes the encoding and decoding functions in the Rust library developed in [189]. This library has been optimized based on low-level benchmarks of binary code's fundamental coding atom—i.e., the $\texttt{xor}_b$ function introduced in Sec. 6.2—and the

[3] `https://www.analog.com/en/products/ltc2991.html` (accessed January 23rd 2024)
[4] `https://store.arduino.cc/products/arduino-nano` (accessed January 23rd 2024)
[5] `https://iperf.fr/` (accessed January 23rd 2024)

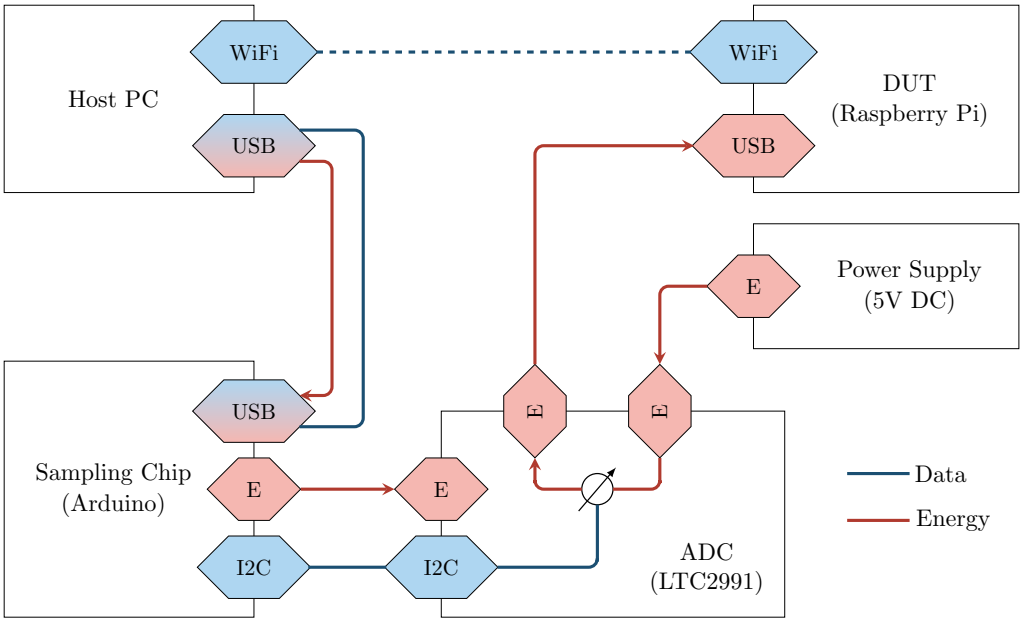


Figure 6.10: Hardware setup for the energy measurements. Reproduced from [179].

custom binary matrix. The custom binary matrix avoids bound checks at execution time when iterating through all elements—these checks occur at compilation time instead. As a result, the matrix performs faster lookups than other existing Rust crates, thereby outperforming them in the execution of the MP algorithm. According to [189], the best performance on the Raspberry Pi Zero W is achieved by $\texttt{xor}_{64}$ and the `u32`-based binary matrix.

The communication command takes as input the number of packets to transmit or receive, the two code commands take the block length and number of parity packets as inputs. The output of the commands is the time it takes to execute the operation in milliseconds. Fig. 6.11 depicts an energy measurement example for some of these commands. Every piece of code is iterated multiple times in order to produce a perceivable power draw that can be isolated from the background power the DUT draws when idle. For example, MDS en-/decode operations are iterated 500 times, packet transmission and reception 1,000 times, and random fountain en-/decode 3,000 times. Measurements have been collected for 25 executions of each command, grouped in sets of 5 executions with a sleep time of 5 seconds between executions in the case of MDS and random fountain codes, and a sleep time of 15 seconds in the transmission due to the noisier measurements obtained in this case.

A post-processing step is applied to the raw measurements from the setup to extract the power draw resulting from the command execution. The background power draw

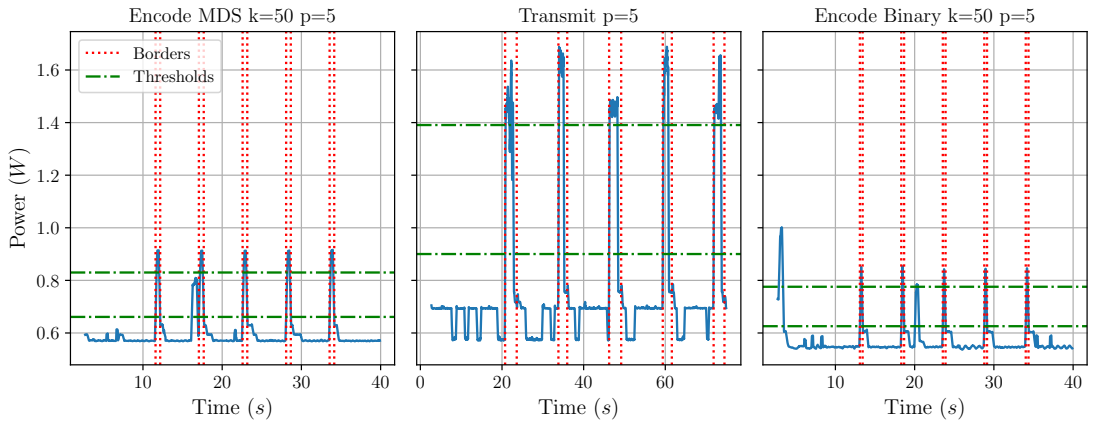


Figure 6.11: Automatic detection of the power draw caused by data transmission and block encoding with MDS and binary codes.

(*background*) is measured at the beginning of the experiment when no code is executed. The upper ($threshold_{up}$) and lower ($threshold_{low}$) thresholds are obtained as follows:

$$threshold_{up} = \frac{max + middle}{2} \tag{6.37}$$

$$threshold_{low} = \frac{background + middle}{2} \tag{6.38}$$

where max is the maximum measured power draw, and $middle = \frac{1}{2} \cdot (background + max)$ is the middle point between the background and maximum power. The script searches for the code execution only within the two thresholds and uses the timing information output by the commands to find the two borders around the code execution depicted in Fig. 6.11. This code detection mechanism avoids misclassification—e.g., see the power draw peak before the second execution in the MDS figure. Once the borders are found, the host PC obtains the average energy demand per iteration dividing the power draw between the borders by the sampling period—i.e., 4ms—and the aforementioned number of iterations per command.

Model Construction

Sec. 6.1 shows that the matrix-vector multiplication dominates the en-/decoding complexity in the transport layer. For a fixed packet length and number of parity packets to en-/decode, Eq. (6.4), Eq. (6.5), Eq. (6.8), and Eq. (6.9) show that the number of operations grows linearly with the block length.[6] Given $\mathcal{X}$ and $\mathcal{Y}$ a set of N samples generated with the linear function $f : X \to Y$. Linear Least Squares (LLS) enables the

[6] In the decoding case, the behavior is superlinear due to the matrix inversion component, but the results presented in this section show that linear estimation still achieves a low estimation error nonetheless.

construction of an estimator of the function f that minimizes the sum of squared errors as follows

$$\hat{y} = m \cdot x + b \tag{6.39}$$

$$m = \frac{N \cdot \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} xy - \sum_{x \in \mathcal{X}} x \sum_{y \mathcal{Y}} y}{N \cdot \sum_{x \in \mathcal{X}} x^2 - (\sum_{x \in \mathcal{X}} x)^2} \tag{6.40}$$

$$b = \frac{1}{N} \cdot \left(\sum_{y \in \mathcal{Y}} y - m \cdot \sum_{x \in \mathcal{X}} x \right) \tag{6.41}$$

In the particular case of energy demand in HARQ, $x \in \mathbb{R}$ and $y \in \mathbb{Q}$. Eq. (6.35) introduced four different energy models that are required in HARQ. Eq. (6.42) and Eq. (6.43) show how these models have been constructed with LLS. Table 6.2 collects the parameters β_1 and β_0.

$$E_{en/decoding}(k, p) = p \cdot (\beta_{1 en/decoding} \cdot k + \beta_{0 en/decoding}) \tag{6.42}$$

$$E_{Tx/Rx}(p) = \beta_{1 Tx/Rx} \cdot p + \beta_{0 Tx/Rx} \tag{6.43}$$

Table 6.2: Linear functions to obtain the energy demand in milliJules

Function	Description	β_1	β_0
$E^{MDS}_{encoding}(k, p)$	MDS Encoding	0.0065578	-0.0168258
$E^{MDS}_{decoding}(k, p)$	MDS Decoding	0.0062807	-0.0146655
$E^{Bin}_{encoding}(k, p)$	Binary Encoding	0.0006568	-0.0026564
$E^{Bin}_{decoding}(k, p)$	Binary Decoding	0.000638	-0.002844
$E_{Tx}(p)$	Data Transmission	0.3315099	0.0937526
$E_{Rx}(p)$	Data Reception	0.0472131	0.007368

Fig. 6.12 compares the estimated energy demand of MDS and random fountain binary codes with the measured values. For every k and p, 25 measurements have been performed in batches of 5 (see the methodology above). The error bars in the measured lines in the standard deviation measured in the 25 executions. The estimation error is within the standard deviation in all cases, thereby proving the robustness of LLS for the model construction. Fig. 6.13 shows the same model robustness for the energy demand of the data transmission and reception.

6.4.2 Delay Model

The energy demand difference between MDS and random fountain codes stems from the time they take to en-/decode the parity packets. The faster coding in random fountain

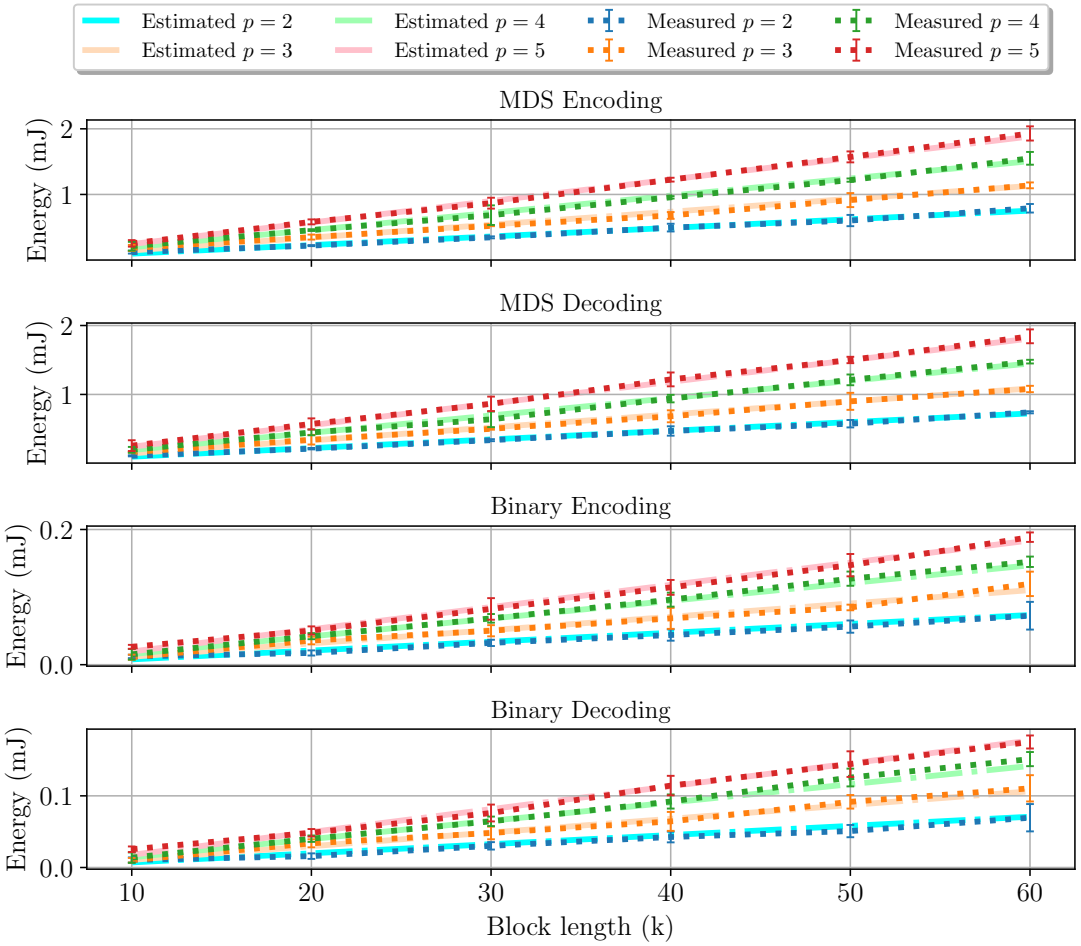


Figure 6.12: En-/decoding energy demand for MDS and random fountain codes on a Raspberry Pi Zero W. *Measured* lines correspond to the average energy demand with the standard deviation as error lines.

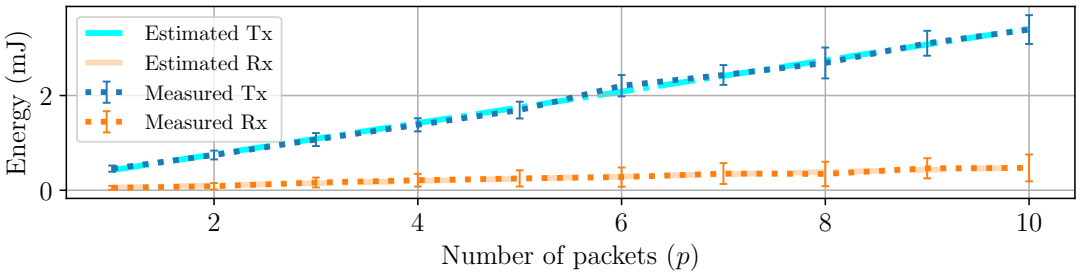


Figure 6.13: Packet transmission and reception energy demand on a Raspberry Pi Zero W. *Measured* lines correspond to the average energy demand with the standard deviation as error lines.

codes results in fewer CPU cycles per parity packet than MDS-generated parity packets, thereby reducing its power draw. Similarly to the energy models for the Raspberry Pi Zero W presented above, delay models for the encoding and decoding operations can also be generated (see Eq. (6.44) and Table 6.3).

$$D_{en/decoding}(k,p) = p \cdot (m_{en/decoding} \cdot k + b_{en/decoding}) \tag{6.44}$$

Table 6.3: Linear functions to obtain the delay in milliseconds.

Function	Description	β_1	β_0
$D^{MDS}_{encoding}(k,p)$	MDS Encoding	0.0190005	-0.0216464
$D^{MDS}_{decoding}(k,p)$	MDS Decoding	0.0183721	-0.0212933
$D^{Bin}_{encoding}(k,p)$	Binary Encoding	0.0020142	-0.0038370
$D^{Bin}_{decoding}(k,p)$	Binary Decoding	0.0019384	-0.0046915

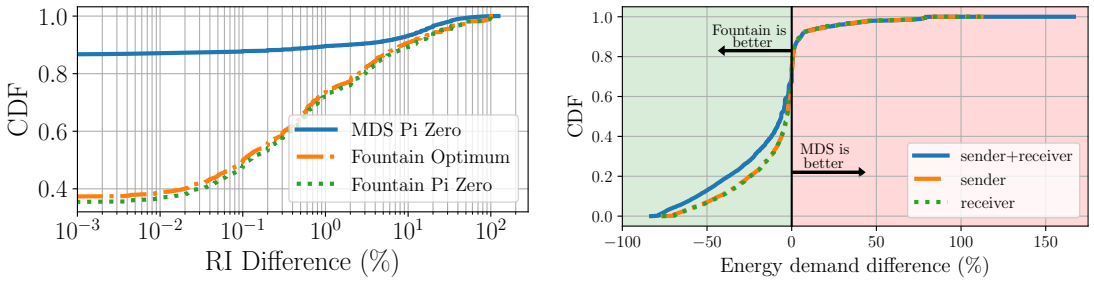
$$D^{\text{Pi Zero}}_{HARQ}(k,N_C,N_P) = D_{FEC}(k,N_P) + D_{ARQ}(N_C,N_P) - \frac{RTT}{2} + D^{\text{Pi Zero}}_{MDS/Bin}(k,p) \tag{6.45}$$

$$D^{\text{Pi Zero}}_{MDS/Bin}(k,p) = D^{MDS/Bin}_{encoding}(k,p) + D^{MDS/Bin}_{decoding}(k,p) \tag{6.46}$$

The delay model in the SHARQ and DeepSHARQ algorithms assumes that parity packets can be en-/decoded infinitely fast (see Eq. (3.16)). This assumption is based on the en-/decoding delay on high-end desktop PCs, which is in the order of tens to hundreds of microseconds. Such delays are one to two orders of magnitude smaller than the delay target of the most stringent applications, namely 1ms (see Table 3.2). However, as PRRT is brought to embedded devices such as the Raspberry Pi Zero W, these delays become non-negligible. The delay budget for the Raspberry Pi Zero W is shown in Eq. (6.45), where $D^{\text{Pi Zero}}_{MDS/Bin}(k,p)$ is the coding delay model for either MDS or random fountain codes.

6.5 Practical Analysis

As the parity packet en-/decoding delay becomes larger, it takes a more significant portion of the available delay budget. Consequently, the available time for the HARQ scheme to recover packet losses is inversely proportional to the en-/decoding delay. This reduction of the effective time budget is expected to produce an RI increase from the optimum found for MDS codes when the en-/decoding delays are zero. Fig. 6.14a depicts the RI increase distribution for i) MDS codes running on the Raspberry Pi Zero W (*MDS Pi Zero*), ii) random fountain codes with parity packets being en-/decoded infinitely fast



(a) **(b)**

Figure 6.14: (a) RI increase when en-/decoding delay takes a portion of the delay budget and/or excess parity packets are required. (b) CDF of the energy demand difference between random fountain and MDS codes.

(*Fountain Optimum*), and iii) random fountain codes running on the Raspberry Pi Zero W (*Fountain Pi Zero*). When the code processing delay is included in the delay budget, MDS and random fountain codes introduce no RI increase in 86.6% and 35.24% of the cases, respectively. More configurations experience an RI increase with random fountain codes as a result of the excess parity packets. Nevertheless, the RI increase is kept low for a significant number of cases, e.g., less than 1% and 10% in 72% and 89% of the cases, respectively. Despite the excess parity packets they require, the tail RI increase is smaller for random fountain codes (106.38%) than for MDS codes (123.65%). This result is explained by the smaller portion of the delay budget binary codes take to en-/decode—e.g., encoding 2 parity packets with a block length $k = 20$ takes approximately 700µs with MDS codes and 70µs with random fountain codes.

The higher efficiency of MDS codes in terms of transmitted RI is not reflected in their energy efficiency. Fig. 6.14b depicts the CDF of the energy demand difference between random fountain and MDS. A negative percentage means that the fountain code demands less energy than MDS, and vice versa. Three different configurations have been depicted: i) *sender+receiver* assumes that both the sender and the receiver in the communication are Raspberry Pi Zero W, ii) in *sender* only the sender is a Raspberry Pi Zero W, and iii) in *receiver* only the receiver is a Raspberry Pi Zero W. The latter two simulate systems with resource-constrained embedded systems in one of the two ends of the communication, and hence the other end is assumed to be able to en-/decode infinitely fast. Table 6.4 complements the figure with the exact percentages of samples in which each code outperforms the other. Even in their worst-case scenario (*sender*) the random fountain codes demand less or the same amount of energy in approximately 75% of all cases. The most extreme differences are found in the sender+receiver case, with the lowest difference at -83% and the largest one at 166.6%. Nevertheless, fountain codes introduce an increase of 50% and 100% to less than 2% and 0.1% of the samples, respectively. Although random fountain codes demand less energy than MDS codes in 63% of the cases, the latter are better in 15% of the cases, which is a significant portion

Table 6.4: Percentage of samples in which the random fountain code demands less, the same, or more energy than an MDS code when both run on a Raspberry Pi Zero W.

Configuration	$E^{Bin}_{HARQ} < E^{MDS}_{HARQ}$	$E^{Bin}_{HARQ} = E^{MDS}_{HARQ}$	$E^{Bin}_{HARQ} > E^{MDS}_{HARQ}$
sender+receiver	62.219%	15.300%	22.479%
sender	59.937%	15.342%	24.720%
receiver	60.361%	15.283%	24.355%

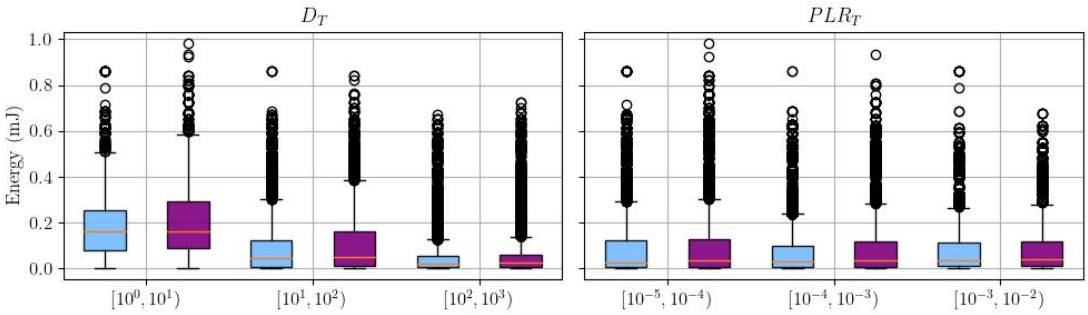


Figure 6.15: Total energy demand of random fountain codes (blue) and MDS codes (purple) on the Raspberry Pi Zero W split by orders of magnitude in the D_T and PLR_T constraints. Both the sender and receiver are assumed to run on a Raspberry Pi Zero W for the results depicted here.

of the dataset for MDS to be neglected. As predicted in the theoretical analysis in Sec. 6.3, there is no one-size-fits-all code when it comes to implementing an energy-aware, predictably reliable, real-time transport service.

Fig. 6.15 depicts the distribution of the total energy demand split by different ranges of the target delay (D_T) and packet loss rate (PLR_T) constraints. Scenarios operating with a small time budget tend to favor proactive cycles due to the lack of time for receiver feedback collection, which explains the inversely proportional relation between E_{HARQ} and D_T depicted on the left figure in Fig. 6.15. Therefore, binary codes seem a more suitable option for applications with stringent delay constraints. The difference between random fountain and MDS codes diminishes as D_T increases. This behavior could be explained by the fact that the relative portion of the time budget spent on parity packet en-/decoding is lower in this case. Unlike the distributions for the target delay, the PLR_T-sorted distributions seem more homogeneous for the different ranges. This result may seem counter-intuitive at first sight, since binary codes have a higher error floor (see Sec. 6.1.2) that would hinder their performance for the low target loss rates. However, in an HARQ scheme, the contribution of excess packets is weighted by the probability of their cycle being triggered (see Eq. (6.36)). The optimal repair schedule construction

in Sec. 4.3.2 minimizes their contribution to the total energy demand of a configuration, whereas the simple schedule in Sec. 5.2.3 puts them in the last cycle.

6.6 Discussion

This section discusses the theoretical and practical evaluations of the complexity dilemma that emerges with the introduction of binary codes.

6.6.1 Complexity Dilemma

The complexity dilemma in error coding takes a completely different shape when observed from the perspective of delay-constrained communication. The matrix inversion has been the main target in every new code proposal for the physical layer, which plays little to no role when a code is used in packetized layers (see Sec. 6.1.3). Reducing the average column degree decreases the number of operations required to generate a parity packet. However, Sec. 6.3.1 has shown that the impact of the CPU's bit-width plays a more important role in the transport layer. The MDS code implemented in PRRT processes the packets byte-by-byte since it operates in $GF(2^8)$. The number of bits per symbol could be increased by using a higher order field, but that would increase the size of the tables employed to reduce the complexity of the field multiplication (see Sec. 3.3.3 for more details). On the contrary, binary codes perform the matrix-vector multiplication with plain XOR operations, so that the number of bits grouped in a single instruction does not depend on the underlying field, but the bit-width instead. On the one hand, the average column degree in random fountain codes is $\frac{k}{2}$, which halves the number of operations per column in comparison to MDS. On the other hand, XORing two packets requires 4 or 8 fewer instructions on 32-bit and 64-bit CPUs, respectively. The higher energy efficiency of random fountain codes shown in Sec. 6.5 is a result of the combined effect of these two complexity reduction sources.

6.6.2 Delay Budget

Previous models for delay-constrained communication have ignored the processing delay for parity packet en-/decoding (see Sec. 3.5.1, which is based on [57, 86]), as they assume that parity packets can be generated infinitely quickly. This assumption is based on en-/decoding delays on desktop PCs and servers, typically several orders of magnitude smaller than other delay sources such as the propagation delay. Not only is the low delay a result of higher CPU clock speed, but of features that are not typically supported on low-end devices, e.g., specific CPU instructions to operate on high-order Galois Fields, multiple CPU cores [52], SIMD instructions [61], or even GPUs [62].

As transport protocols are brought to resource-constrained embedded devices, the natural component of CPSs, the en-/decoding delay becomes non-negligible and can take a significant portion of the delay budget, especially for applications with the most stringent delay requirements. Therefore, the available time budget for erasure coding is effectively reduced. Sec. 6.5 has shown that this budget reduction results in an average

RI increase of 10% in MDS codes when compared to the optimum, device-independent RI. However, MDS and random fountain codes introduce no RI increase in 86.6% and 35.24% of the cases, respectively. In other words, from a purely information-theoretical standpoint, MDS codes remain the best option on average. This statement does not hold true for all samples though, since Fig. 6.14a has shown that random fountain codes have a smaller tail RI increase than MDS. Nevertheless, this performance inversion is negligible due to the small number of samples for which it occurs. Since the effective time budget is inversely proportional to the code complexity, Sec. 6.5 has also shown that the more efficient binary codes introduce an average RI increase of only 1%, despite the parity packet excess introduced due to the lack of generator matrix invertibility guarantees. These results suggest that binary codes are a better candidate as delay constraints become tighter and/or the PRRT protocol is deployed on low-end devices with fewer computational resources.

6.6.3 Energy Awareness

The systematic analysis of the energy demand of the coding function in HARQ here presented is a first step towards a truly energy-aware transport protocol that brings predictably reliable, real-time transport to embedded devices. The results suggest there is not a fit-all code that outperforms the others in terms of energy efficiency in all scenarios. However, the myriad of codes in the literature [15, 23, 26, 32, 40, 45, 60, 123, 129] makes it unfeasible to extend the same analysis to every code with their different complexity and energy footprint, not to mention that the energy models are completely platform-dependent. The intractability of the impact of the different system variables on its energy demand is a well-known problem in the energy-aware community [111, 190]. Providing developers with the expected energy demand of a code on different platforms [88, 97] or automatically constructing energy models based on the powerful feature extraction of machine learning [153] are some of the solutions that have been applied in that field. Little attention has been paid to the application of these energy-aware solutions to transport protocols, but it seems a promising research field that will attract attention as transport protocols are deployed on constrained devices [176]. The modular design of transport protocols [148] creates the opportunity to perform energy-aware selections of the transport layer functions based on the scenario they will be deployed on. DL-based heuristics could play an important role in enabling such a dynamic protocol configuration, as they can be designed to achieve good performance for different target metrics and application scenarios [210]. For example, the DeepSHARQ algorithm in Chapter 5 could be retrained with transfer learning [46, 73] in order to support a different code—e.g., random fountain—or a new target metric—e.g., energy demand.

6.6.4 Other Codes

This chapter has only considered MDS and binary block codes. However, Random Linear Network Coding (RLNC) codes [43] have also been implemented in packetized

layers, both in the block [72, 92, 132, 211] and stream [129, 150, 188, 206] form. RLNC codes were originally proposed in the context of what is called the *network information flow problem* [29] in multisource multicast networks. Ahlswede et al. [29] show that bandwidth can be saved in this scenario by employing coding at the network nodes instead of applying solely routing and forwarding as IP routers do. An in-depth analysis of this problem is out of the scope of this thesis, and the remainder of this section focuses on the suitability of RLNC codes for delay-constrained communication on embedded devices.

RLNC follows a similar philosophy to random fountain codes: the elements of the generator matrix are randomly selected from an underlying field. However, in order to increase the probability of obtaining linearly independent columns, they employ higher-order fields instead of $GF(2)$. Ho et al. [43] prove that these codes are capacity approaching with probability exponentially approaching 1 with the block length. Shojania et al. [52, 61, 62] bring RLNC to low-end devices by employing parallel computing—i.e., multithreading [52] and GPUs [62]—to reduce the computational complexity of the matrix inversion, as these codes operate in the large block length regime. [72] shows that RLNC codes achieve a lower en-/decoding throughput than binary codes due to their higher complexity. It is precisely their high complexity that prevents them from being a performant option for delay-constrained applications on embedded devices. Since GPUs are becoming more common on embedded devices due to the broad adoption of TinyML (see Sec. 5.1.2), it is an open question as to whether RLNC codes will be viable in this setup in the near future as energy-efficient GPUs are released.

Stream codes make different use of the time budget than the block codes employed by PRRT. In a delay-constrained block code, only the full budget of the first data packet in the block is fully employed: PRRT optimizes the HARQ scheme so that the first packet in the block is never delivered to the application later than its target delay. As a result, there is a residual delay budget for later data packets in the block left unused, even though there would still be time left for those packets to be recovered. On the contrary, stream codes [129, 150] make the end-to-end delay independent of the block length by evenly spreading the parity packets over the source data packets. Stream codes can generate parity packets at any time by computing a linear combination of the source packets within a D_T-dependent sliding window. Fong et al. [150] show the complexity of creating a windowed generator matrix whose invertibility is not guaranteed for two reasons: i) the random nature of the matrix elements does not ensure the columns are linearly independent, and ii) the cross dependencies between different windows. Therefore, although they make a more efficient use of the time budget, stream codes are suboptimal from the matrix construction point of view.

6.7 Related Work

Two main trends have emerged in the last two decades when it comes to energy awareness in packetized protocols—i.e., layers 2 to 4 of the protocol stack—, namely sustainable large-scale deployments [34, 216, 220] and energy-efficient protocols for low-end

devices [33, 35, 37, 39, 49, 51, 176, 204]. Gupta et al. [34] perform an extensive analysis of the energy demand of Internet devices, and proposed changes in the network architecture to increase the average sleeping time of network devices. Fotino et al. [49] reach a similar conclusion after analyzing the energy demand of routing algorithms in ad hoc networks built by the interconnection of low-end devices. Two decades later, [220] and [216] extend the focus of the sustainability debate to introduce CO_2 emissions to the analysis as well, so that the usage of low-emissions energy sources is favored. Jacob et al. [220] propose still longer sleeping times for network devices as one of the major vectors when it comes to energy efficiency, which should be combined with an extended device lifespan to reduce the Internet's carbon footprint. Zilberman et al. [216] go a step further in their proposal by exploiting network telemetry and programmable network devices to implement a carbon-aware Internet that selects routes based on their carbon footprint.

To the best of our knowledge, [33] is the first full IP/TCP stack implementation targeted at resource-constrained devices. Code size, effective RAM usage, and communication performance in terms of throughput and delay are the main focuses of the article, but energy efficiency is out of its scope. Wan et al. [37] implement an energy-efficient congestion control algorithm for event-driven sensor networks, which configures a channel sampling rate at the receiver that reduces the sampling time and the energy demand it entails, and applies backpreasure to avoid congestion. Since full reliability increases the energy demand when the same event is detected by multiple receivers, which in turn communicate it to a central node, [35] implements another receiver-based congestion detection providing event detection reliability instead of packet-level reliability. Iyer et al. [39] offload the end nodes by deploying the transport layer functions, such as partial reliability, and congestion detection and avoidance, on the base stations. Similarly to PRRT, the proposed protocol supports different per-flow reliability levels, and NAKs are only triggered if a flow reliability level is below the target. TinyNet [204] is a lightweight protocol stack tailored for low-power networks. The authors report that TinyNet has a low memory footprint, and it outperforms similar stacks in terms of delay and energy efficiency. Kumar et al. [176] show that off-the-shelf implementations of TCP achieve poor performance on IoT deployments. The authors systematically analyze the reasons for such a poor performance—i.e., small packet size, hidden-node problem, and unsynchronized layer 2 and 4 protocols—and provide a custom adaptation of an existing TCP stack that overcomes them.

Most similar to the analysis presented in this chapter are, to the best of our knowledge, [61, 62, 72, 132]. Shojania et al. [61] provide a systematic performance analysis of RLNC codes in an IoT deployment. The target platform is an iPhone 3G, which has the same ARMv6 CPU architecture as the Raspberry Pi Zero W. The major difference from the analysis presented in this chapter is the block length regime, as [61] focuses on $k = 4096$ for which the matrix-vector multiplication implemented in PRRT (see Sec. 3.1.2) is unfeasible due to the size of the tables. As a result, the authors have opted for an algorithm that leverages parallel processing with either multithreading in [52] or GPUs in [62]. The Tenor platform in [72] supports LT, RLNC and MDS codes optimized

for ARM platforms, and shows that the binary LT codes achieve the higher en-/decoding throughput on constrained devices. Finally, Wunderlich et al. [132] bring RLNC codes to IoT devices with big.LITTLE multicore architecture with multiple big—i.e., fast but power-hungry—cores and multiple LITTLE—i.e., slow but power-efficient—cores.

6.8 Conclusion

In the large block length regime, sparse binary codes have a low computational complexity and their excess symbols become negligible. Hence, they have become the de facto standard in the lower layers of the stack. In the short block length regime that is common in delay-constrained transport, sparse codes become unfeasible due to the significant amount of resources that are devoted to transmitting excess packets. This chapter has shown that binary codes are nonetheless an energy-efficient alternative to MDS codes on resource-constrained embedded devices, so long as the number of excess packets is kept low. The non-sparse random fountain and polar codes emerge as an alternative to bring predictable reliability, and delay-constrained transport to devices with limited computational resources. Random fountain codes can reduce the energy demand by 83%, outperforming MDS codes in 59-63% of the cases. However, in a smaller percentage of cases, they introduce an energy demand excess that can reach up to a 166% increase. Therefore, due to the lack of a one-size-fits-all code, an energy-aware HARQ scheme is required to optimize the energy demand depending on the scenario and application requirements. In particular, the results presented here suggest that binary codes are an efficient alternative for applications with the most stringent delay constraints when they are deployed on low-end devices with limited computational and energy resources. This chapter has laid the foundations for the systematic design of these energy-aware HARQ schemes, a new research track that should be further investigated in future work.

Chapter 7

Conclusion

Cyber-Physical Systems (CPSs) bridge the gap between the digital and physical world by closing control loops over embedded computers and networks. Reliability and timeliness are key functional requirements for the correct operation of these systems to ensure a safe operation and the stability of the control loops. As CPSs are expected to permeate our societies, the non-functional requirement of energy efficiency must be considered to ensure a sustainable deployment that has the potential to enable unprecedented efficiency levels in almost any human process to tackle climate change. For CPSs to unfold all their potential, they should leverage open operating systems and protocol stacks providing performance guarantees off the shelf, thereby enabling an ecosystem for developers to focus on the application solely. The PRRT protocol provides a predictably reliable, real-time service to deliver the data to the receiving end within the delay budget and with the reliability level specified by the application. PRRT is a self-aware protocol that monitors self- and channel-induced delays and packet losses to meet the application requirements. The protocol also leverages interfaces with the operating system for precise time and energy-aware decisions. These properties make PRRT an ideal candidate to enable data communication for CPSs.

7.1 Summary

In answering the research question *how should transport protocols be designed to provide broad support for Cyber-Physical Systems?*, this thesis has made original contributions to the design of predictably reliable, delay-aware error control. The main conclusions obtained while performing the research presented here are summarized in the following:

- The data-driven analysis presented in Chapter 3 has shown that even when made delay- and reliability-aware, a pure ARQ parameter search fails to obtain valid configurations meeting the constraints on a wide range of network conditions, while a pure FEC search introduces large RI increases from the optimum. The results prove the need for adaptive HARQ in the transport layer in order to support CPSs

on a wide range of conditions while approaching the channel capacity, thereby optimizing the transmitted RI.

- The computational complexity analysis of the HARQ scheme implemented in PRRT points to the error control function, and more precisely the optimizer and coder components, as the major limiting factor for PRRT to be brought to resource-constrained platforms, i.e., the natural components of CPSs. Currently, no closed-form expression exists to obtain the optimal HARQ configuration fulfilling the constraints. Chapter 4 has presented algorithmic optimizations for a faster exploration of the solution space and construction of the repair schedule. These complexity optimizations reduce the search algorithms' inference time, allowing PRRT to provide a predictable communication channel on high-end PCs.

- Despite the algorithmic optimizations, the inference time of the search algorithms is still too high to bring predictability to resource-constrained devices. We hypothesize that most of the complexity comes from the quantization of the solution space, which makes the search algorithm sensitive to noise such that small variations in the input produce large variations in the output. Chapter 5 presents a DL-based solution that turns these quantization effects into a feature by allowing neural networks to learn any block length that keeps the RI deviation from the optimum within a configurable margin. We show this solution outperforms purely learning-based approaches in terms of the size of the obtained neural networks, thereby bringing predictable transport to low-end devices for the first time.

- Coding at the transport layer demands a revision of the complexity dilemma that has driven code design in the last decades. Chapter 6 shows that this dilemma fundamentally differs in packetized layers because the matrix inversion no longer dominates the coding complexity, but the matrix-vector multiplication does because it must be iterated over full packets. Based on this finding, we systematically analyzed binary code's ability to bring efficient and predictable reliability to delay-aware communications. Using a purely information theoretical metric such as the RI, MDS codes are optimal as every parity packet can correct one lost packet, and hence, they were considered the best option for this type of communication. However, the results presented in this thesis show that, when looking at the overall resource consumption of the platform, binary codes can be more efficient in a significant portion of network conditions, especially when delay constraints are more stringent.

These findings led to the development of three main components that have been implemented as Rust libraries for their simple integration into existing transport protocols:

1. **SHARQ**: this search algorithm has been designed to provide the optimum HARQ configuration in terms of the RI in predictable time. With a fast schedule and solution space exploration, it can bring predictably reliable, real-time transport to high-end devices.

2. **DeepSHARQ**: this search algorithm combines algorithmic and deep learning components to reduce the inference time of finding HARQ configurations. Despite sacrificing optimality, it maintains the RI deviation within configurable bounds, thereby enabling PRRT on resource-constrained devices.

3. **Packetized binary coder**: a random binary coder implementing a custom matrix object that avoids unnecessary memory checks in the existing Rust crates that hinder performance. This component implements an encoder and decoder operating on packets of configurable length.

7.2 Future Work

Despite the original contributions in the area of predictably reliable, delay- and energy-aware transport presented in this thesis, there is still room for future investigations to enable CPSs over IP networks.

Performance in the Wild

This thesis has performed a data-driven analysis of the error control function. Although the dataset is based on traces collected in the wild, they do not portray PRRT's performance when interacting with other protocols and devices once it is deployed. These interactions have been broadly studied for congestion control in TCP [112, 147, 169]. However, TCP's operation point fundamentally differs from PRRT's: while TCP aims to fill the channel to maximize throughput, PRRT must operate below this point to allow for the transmission of parity packets. Similar large-scale experiments should be performed with PRRT, extending the analysis to error control. Particularly relevant would be the evaluations on top of TSN and 5G testbeds, which are the main candidates for wired and wireless CPSs, respectively.

Moreover, cross-layer effects have already been encountered for other real-time applications [1, 89]. Cross-layer effects prevent the application from obtaining the intended performance due to diminishing effects between heuristics in the transport and application layers. Similar effects may occur when control loops are deployed on top of PRRT. Ideally, PRRT could be integrated into some control algorithm performance tool such as [209] so that the impact of cross-layer effects on control performance could be studied.

Channel Estimation

While this thesis has focused on error control, channel estimation is equally relevant for transport predictability. As requirements become more stringent, high-precision channel models will be required to cope with the increased demands for performance guarantees. LSTM-based channel estimation has already achieved good performance when it comes to predicting future channel behavior [146, 171, 191]. An interesting avenue for research would be to leverage information from the lower layers obtained from the large-scale evaluations mentioned above to feed LSTM models.

Energy-Aware Adaptation

The energy-awareness results presented in this thesis show the complexity of the problem at hand: picking the optimal coding technique is a multifactorial problem depending on i) the network conditions, ii) the application requirements, iii) the platform the protocol operates on, and iv) the set of available coding techniques. The DeepSHARQ algorithm can be easily adapted to make device-aware decisions and/or support a different coding technique: transfer learning allows a model to be retrained to fit a slightly different set without going through a complete training session. For example, the new dataset should reflect the optimal configurations for a different code or restrict the supported block lengths due to the coding delays introduced by the device. However, if no code outperforms the others in all circumstances, when to select the coding technique remains an open research question.

The concept of *pluginizing* transport protocols to enable the fast evolution of its internal functions has been proposed by several authors [142, 148]. Different coding mechanisms could be provided as plugins—e.g., using Rust crates and features for conditional compilation. The plugin mechanism would require a heuristic at compilation time that selects the optimal coding technique based on, e.g., the platform or application the protocol is expected to operate with. Alternatively, the selection of the coding mechanism could be integrated into the adaptive HARQ optimization heuristics, which would increase the optimization problem's complexity.

Appendix A

Computational Complexity Packet Loss Rate

The run-time complexity of the HARQ packet loss rate in Eq. (3.17) can be reduced to linear complexity if some optimizations are taken into account. Splitting the PLR into the two components in Eq. (A.1), it can be shown that each of the components can be calculated in $\mathcal{O}(k_{max} + log(p_{max}))$ (see Eq. (A.2) and Eq. (A.4)). Bear in mind that the derivations here presented do consider the alternative expression for the probability of i packet losses in systematic MDS codes presented in Eq. (A.5).

$$PLR_{HARQ}(k,p) = \underbrace{\frac{1}{k}\sum_{i=1}^{min(k,p)} i \cdot Pr(I_k = i)}_{PLR_1(k,p)} + \underbrace{\frac{1}{k}\sum_{i=p+1}^{k} i \cdot Pr(I_k = i)}_{PLR_2(k,p)} \tag{A.1}$$

$$PLR_1(k,p) = \frac{1}{k} \cdot {p_e}^p (1-p_e)^k \cdot \sum_{i=1}^{min(k,p)} i \cdot \binom{k}{i} \cdot PS_p(p,i) \tag{A.2}$$

$$PS_p(p,i) = \frac{p_e}{1-p_e}\left(\binom{p}{i} + PS_p(p,i-1)\right) \tag{A.3}$$

$$PLR_2(k,p) = \frac{1}{k}\sum_{i=p+1}^{k} i \cdot \binom{k}{i} {p_e}^i (1-p_e)^{k-i} \tag{A.4}$$

$$Pr(I_k = i) = \begin{cases} \binom{k}{i} {p_e}^i (1-p_e)^{k-i} & i > p \\ \binom{k}{i} \sum_{f=1}^{i} \binom{p}{p-i+f} {p_e}^{p+f} (1-p_e)^{k-f} & i \leq p \end{cases} \tag{A.5}$$

The complete algorithm to obtain the PLR is shown in Alg. 7, where $PS_P(p,0) = 0$. BINOM is a 256×256 table computed at compilation time so that any $\binom{n}{k}$ can be obtained in $\mathcal{O}(1)$.

Algorithm 7 PLR

Require: k, p, p_e, BINOM
Ensure: $PLR = \frac{1}{k}\sum_{i=1}^{k} i \cdot Pr(I_k = i)$

$ps_p \leftarrow 0$
$plr_1 \leftarrow 0$
for $i = 0 \rightarrow min(k,p)$ **do**
    $plr_1 \leftarrow plr_1 + i \cdot \text{BINOM}[k][i] \cdot ps_p$
    $ps_p \leftarrow \frac{p_e}{1-p_e} \cdot \big(\text{BINOM}[p][p-i] + ps_p\big)$
end for
$plr_1 \leftarrow plr_1 \cdot {p_e}^p \cdot (1-p_e)^k$
$plr_2 \leftarrow 0$
if $p \leq k$ **then**
    $pow \leftarrow {p_e}^p \cdot (1-p_e)^{k-p}$
    for $j = p+1 \rightarrow k$ **do**
        $pow \leftarrow pow \cdot \frac{p_e}{1-p_e}$
        $plr_2 \leftarrow plr_2 + j \cdot \text{BINOM}[k][j] \cdot pow$
    end for
end if
return $\frac{(plr_1+plr_2)}{k}$

Appendix B

Computational Complexity Redundancy Information

The RI expression, originally defined in Eq. (3.11), can be reformulated as shown in Eq. (B.1), where $w(j)$ is the probability that the code $\mathcal{C}(n,k)$ fails after the transmission of $j \leq p = n - k$ parity packets—see Eq. (B.2). Binomial coefficients fulfill that $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-k}{k}$ for $1 \leq k < n$ Following this property of binomial coefficients, the recursive expression for $w(j)$ in Eq. (B.4) can be obtained. The run-time complexity of evaluating Eq. (3.11) with Alg. 8 is $\mathcal{O}(n)$, where BINOM is a 256×256 table computed at compilation time so that any $\binom{n}{k}$ can be obtained in $\mathcal{O}(1)$.

$$RI(k, N_C, N_P) = \frac{1}{k} N_P 0 + \frac{1}{k} \sum_{c=1}^{N_C} w^R\big(\mathbf{n}c - 1 - k\big) \cdot N_P c \tag{B.1}$$

$$w(j) = \sum_{i=max(0,k-p+j)}^{k-1} \binom{k+j}{i} (1-p_e)^i {p_e}^{k+j-i} \tag{B.2}$$

$$w^R(j) = 1 - (1 - w(j))^R \tag{B.3}$$

$$w(j) = \begin{cases} p_e\big(w(j-1) - a_1^1(j)\big) + (1-p_e)\big(w(j-1) - a_2(j)\big) + {p_e}^{k+j} & \text{if } k-p+j < 0 \\ p_e\big(w(j-1) - a_1^2(j)\big) + (1-p_e)\big(w(j-1) - a_2(j)\big) & \text{if } k-p+j \geq 0 \end{cases} \tag{B.4}$$

$$a_1^1(j) = {p_e}^{k+j-1} \tag{B.5}$$

$$a_1^2(j) = \binom{k+j-1}{k+j-p-1} (1-p_e)^{k+j-p-1} {p_e}^p \tag{B.6}$$

$$a_2(j) = \binom{k+j-1}{k-1} (1-p_e)^{k-1} {p_e}^j \tag{B.7}$$

Algorithm 8 RI

Require: k, p, p_e, BINOM
Ensure: w is a vector with $w[j] = \sum_{i=max(0,k-p+j)}^{k-1} \binom{k+j}{i} (1-p_e)^i {p_e}^{k+j-i}$

1: $w \leftarrow zeroes(p+1)$
2: $m \leftarrow max(0, k-p)$
3: $a \leftarrow (1-p_e)^m {p_e}^{k-m}$
4: **for** $j = m \rightarrow k-1$ **do**
5: $\quad w[0] \leftarrow w[0] + \text{BINOM}[k][j] \cdot a$
6: $\quad a \leftarrow a \cdot \frac{1-p_e}{p_e}$
7: **end for**
8: $a_{first} \leftarrow (1-p_e)^m {p_e}^{k-m}$
9: $a_{last} \leftarrow (1-p_e)^{k-1} p_e$
10: **for** $j = 1 \rightarrow p-1$ **do**
11: $\quad$ **if** $k-p+j-1 \geq 0$ **then**
12: $\quad\quad a_1 \leftarrow \text{BINOM}[k+j-1][k-p+j-1] \cdot a_{first}$
13: $\quad\quad a_{first} \leftarrow a_{first} \cdot (1-p_e)$
14: $\quad$ **else**
15: $\quad\quad a_1 \leftarrow a_{first}$
16: $\quad\quad a_{first} \leftarrow a_{first} \cdot p_e$
17: $\quad$ **end if**
18: $\quad a_2 \leftarrow \text{BINOM}[k+j-1][k-1] \cdot a_{last}$
19: $\quad a_{last} \leftarrow a_{last} \cdot p_e$
20: $\quad w[j] \leftarrow p_e(w[j-1] - a_1) + (1-p_e)(w[j-1] - a_2)$
21: $\quad$ **if** $k-p+j-1 < 0$ **then**
22: $\quad\quad w[j] \leftarrow w[j] + a_{first}$
23: $\quad$ **end if**
24: **end for**
25: **return** w

Own Publications

[1] P. Gil Pereira, A. Schmidt, and T. Herfet. "Cross-Layer Effects on Training Neural Algorithms for Video Streaming". In: *Proceedings of the 28th ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video*. 2018, pp. 43–48.

[2] P. Gil Pereira and T. Herfet. "Minimal Startup Delay in HAS with Staggered Segments". In: *Proceedings of the 16th IEEE Annual Consumer Communications & Networking Conference*. 2019, pp. 1–6.

[3] A. Schmidt, S. Reif, P. Gil Pereira, T. Honig, T. Herfet, and W. Schröder-Preikschat. "Cross-Layer Pacing for Predictably Low Latency". In: *Proceedings of the 6th IEEE Intl. Workshop on Ultra-Low Latency in Wireless Networks*. 2019, pp. 1–6.

[4] M. Böhmer, A. Schmidt, P. Gil Pereira, and T. Herfet. "Latency-Aware and -Predictable Communication with Open Protocol Stacks for Remote Drone Control". In: *Proceedings of the 2nd IEEE Int. Workshop on Wireless Sensors and Drones in Internet of Things*. 2020, pp. 304–311.

[5] S. Reif, B. Herzog, P. Gil Pereira, A. Schmidt, T. Büttner, T. Hönig, W. Schröder-Preikschat, and T. Herfet. "X-Leep: Leveraging Cross-Layer Pacing for Energy-Efficient Edge Systems". In: *Proceedings of the 1st ACM Workshop on Energy Efficiency at the Edge*. 2020, pp. 548–553.

[6] A. Schmidt, P. Gil Pereira, and T. Herfet. "Predictably Reliable Real-time Transport Services for Wireless Cyber-Physical Systems". In: *Proceedings of the 21st IFAC World Congress*. 2020, pp. 2638–2641.

[7] P. Gil Pereira and T. Herfet. "Reducing FEC-Complexity in Cross-Layer Predictable Data Communication". In: *Proceedings of the 18th IEEE Consumer Communications & Networking Conference (Poster Session)*. 2021, pp. 1–2.

[8] P. Gil Pereira and T. Herfet. "Polar Coding for Efficient Transport Layer Multicast". In: *Proceedings of the 19th IEEE Consumer Communications & Networking Conference*. 2022, pp. 313–318.

[9] P. Gil Pereira, A. Schmidt, and T. Herfet. "DeepHEC: Hybrid Error Coding Using Deep Learning". In: *Proceedings of the 18th IEEE European Dependable Computing Conference*. 2022.

[10] P. Gil Pereira, K. Vogelgesang, M. Miodek, A. Schmidt, and T. Herfet. “DeepSHARQ: Hybrid Error Coding Using Deep Learning”. In: *Journal of Reliable Intelligent Environments* (2023), pp. 1–19.

[11] K. Vogelgesang, P. Gil Pereira, and T. Herfet. “SHARQ: Scheduled HARQ for Time- and Loss-Rate-Sensitive Networks”. In: *Proceedings of the 20th IEEE Consumer Communications & Networking Conference*. 2023, pp. 640–643.

Bibliography

[12] W. S. Jevons. *The Coal Question; An Inquiry Concerning the Progress of the Nation, and the Probable Exhaustion of Our Coal Mines*. Macmillan & Co. London, 1865.

[13] C. E. Shannon. "A Mathematical Theory of Communication". In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423.

[14] N. Wiener. *Cybernetics or Control and Communication in the Animal and the Machine*. MIT press, 1961.

[15] R. Gallager. "Low-Density Parity-Check Codes". In: *IRE Transactions on information theory* 8.1 (1962), pp. 21–28.

[16] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error Correcting Codes*. Elsevier, 1977.

[17] J. Postel. *User Datagram Protocol*. RFC 768. IETF, 1980.

[18] J. H. Saltzer, D. P. Reed, and D. D. Clark. "End-to-End Arguments in System Design". In: *ACM Transactions on Computer Systems* 2.4 (1984), pp. 277–288.

[19] R. W. Farebrother. *Linear Least Squares Computations*. CRC Press, 1988.

[20] K. Hornik, M. Stinchcombe, and H. White. "Multilayer Feedforward Networks are Universal Approximators". In: *Neural networks* 2.5 (1989), pp. 359–366.

[21] E. Ostrom. *Governing the Commons: The Evolution of Institutions for Collective Action*. Cambridge University Press, 1990.

[22] J. Blömer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman. *An XOR-Based Erasure-Resilient Coding Scheme*. Tech. rep. TR-95-048. ICSI, Berkeley, California, 1992.

[23] C. Berrou, A. Glavieux, and P. Thitimajshima. "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes". In: *Proceedings of the 3rd IEEE Int. Conference on Communications*. Vol. 2. 1993, pp. 1064–1070.

[24] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. *TCP Selective Acknowledgment Options*. RFC 2018. IETF, 1996.

[25] S. Bradner. *Key Words for Use in RFCs to Indicate Requirement Levels*. RFC 2119. IETF, 1997.

[26] L. Rizzo. “Effective Erasure Codes for Reliable Computer Communication Protocols”. In: *ACM SIGCOMM Computer Communication Review* 27.2 (1997), pp. 24–36.

[27] W. R. Stevens. *TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms*. RFC 2001. IETF, 1997.

[28] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. “A Digital Fountain Approach to Reliable Distribution of Bulk Data”. In: *ACM SIGCOMM Computer Communication Review* 28.4 (1998), pp. 56–67.

[29] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. “Network Information Flow”. In: *IEEE Transactions on Information Theory* 46.4 (2000), pp. 1204–1216.

[30] G. Bernat, A. Burns, and A. Liamosi. “Weakly Hard Real-Time Systems”. In: *IEEE Transactions on Computers* 50.4 (2001), pp. 308–321.

[31] K. Ramakrishnan, S. Floyd, and D. Black. *The Addition of Explicit Congestion Notification (ECN) to IP*. RFC 3168. IETF, 2001.

[32] M. Luby. “LT Codes”. In: *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science*. 2002, pp. 271–271.

[33] A. Dunkels. “Full TCP/IP for 8-bit Architectures”. In: *Proceedings of the 1st Int. Conference on Mobile Systems, Applications and Services*. 2003, pp. 85–98.

[34] M. Gupta and S. Singh. “Greening of the Internet”. In: *Proceedings of the 17th ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. 2003, pp. 19–26.

[35] Y. Sankarasubramaniam, Ö. B. Akan, and I. F. Akyildiz. “ESRT: Event-to-Sink Reliable Transport in Wireless Sensor Networks”. In: *Proceedings of the 4th ACM Int. Symposium on Mobile Ad Hoc Networking & Computing*. 2003, pp. 177–188.

[36] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550. IETF, 2003.

[37] C.-Y. Wan, S. B. Eisenman, and A. T. Campbell. “CODA: Congestion Detection and Avoidance in Sensor Networks”. In: *Proceedings of the 1st Int. Conference on Embedded Networked Sensor Systems*. 2003, pp. 266–279.

[38] D. A. Patterson. “Latency Lags Bandwith”. In: *Communications of the ACM* 47.10 (2004), pp. 71–75.

[39] Y. G. Iyer, S. Gandham, and S. Venkatesan. “STCP: a Generic Transport Layer Protocol for Wireless Sensor Networks”. In: *Proceedings of the 14th IEEE Int. Conference on Computer Communications and Networks*. 2005, pp. 449–454.

[40] D. J. C. MacKay. “Fountain Codes”. In: *IEE Proceedings-Communications* 152.6 (2005), pp. 1062–1068.

[41] O. Tickoo, V. Subramanian, S. Kalyanaraman, and K. K. Ramakrishnan. "LT-TCP: End-to-End Framework to Improve TCP Performance over Networks with Lossy Channels". In: *Proceedings of the 13th Int. Workshop on Quality of Service*. Springer. 2005, pp. 81–93.

[42] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil. "Model Compression". In: *Proceedings of the 12th ACM SIGKDD Int. Conference on Knowledge Discovery and Data Mining*. 2006, pp. 535–541.

[43] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong. "A Random Linear Network Coding Approach to Multicast". In: *IEEE Transactions on Information Theory* 52.10 (2006), pp. 4413–4430.

[44] E. Kohler, M. Handley, and S. Floyd. *Datagram Congestion Control Protocol (DCCP)*. RFC 4340. IETF, 2006.

[45] A. Shokrollahi. "Raptor Codes". In: *IEEE Transactions on Information Theory* 52.6 (2006), pp. 2551–2567.

[46] M. Asadi and M. Huber. "Effective Control Knowledge Transfer through Learning Skill and Representation Hierarchies". In: *Proceedings of the 16th Int. Joint Conference on Artificial Intelligence*. Vol. 7. 2007, pp. 2054–2059.

[47] A. F. Atiya, S. G. Yoo, K. T. Chong, and H. Kim. "Packet Loss Rate Prediction Using the Sparse Basis Prediction Model". In: *IEEE Transactions on Neural Networks* 18.3 (2007), pp. 950–954.

[48] J. Baillieul and P. J. Antsaklis. "Control and Communication Challenges in Networked Real-Time Systems". In: *Proceedings of the IEEE* 95.1 (2007), pp. 9–28.

[49] M. Fotino, A. Gozzi, F. De Rango, S. Marano, J. C. Cano, C. Calafate, and P. Manzoni. "Evaluating Energy-Aware Behaviour of Proactive and Reactive Routing Protocols for Mobile Ad Hoc Networks". In: *Proceedings of the 10th Int. Symposium on Performance Evaluation of Computer and Telecommunication Systems*. 2007, pp. 16–18.

[50] A. Li. *RTP Payload Format for Generic Forward Error Correction*. RFC 5109. IETF, 2007.

[51] J. Paek and R. Govindan. "RCRT: Rate-Controlled Reliable Transport for Wireless Sensor Networks". In: *Proceedings of the 5th Int. Conference on Embedded Networked Sensor Systems*. 2007, pp. 305–319.

[52] H. Shojania and B. Li. "Parallelized Progressive Network Coding with Hardware Acceleration". In: *Proceedings of the 15th IEEE Int. Workshop on Quality of Service*. 2007, pp. 47–55.

[53] S. Ha, I. Rhee, and L. Xu. "CUBIC: a New TCP-Friendly High-Speed TCP Variant". In: *ACM SIGOPS Operating Systems Review* 42.5 (2008), pp. 64–74.

[54] E. A. Lee. "Cyber Physical Systems: Design Challenges". In: *Proceedings of the 11th IEEE Int. Symposium on Object and Component-Oriented Real-time Distributed Computing*. 2008, pp. 363–369.

[55] S. Murugesan. “Harnessing Green IT: Principles and Practices”. In: *IT Professional* 10.1 (2008), pp. 24–33.

[56] P. Regnier, G. Lima, and L. Barreto. “Evaluation of Interrupt Handling Timeliness in Real-Time Linux Operating Systems”. In: *ACM SIGOPS Operating Systems Review* 42.6 (2008), pp. 52–63.

[57] G. Tan. “Optimum Hybrid Error Correction Scheme under Strict Delay Constraints”. PhD thesis. Saarland Informatics Campus, 2008.

[58] S. Ahmad, R. Hamzaoui, and M. Al-Akaidi. “Adaptive Unicast Video Streaming with Rateless Codes and Feedback”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 20.2 (2009), pp. 275–285.

[59] M. Allman, V. Paxson, and E. Blanton. *TCP Congestion Control*. RFC 5681. IETF, 2009.

[60] E. Arikan. “Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels”. In: *IEEE Transactions on Information Theory* 55.7 (2009), pp. 3051–3073.

[61] H. Shojania and B. Li. “Random Network Coding on the iPhone: Fact or Fiction?” In: *Proceedings of the 18th Int. Workshop on Network and Operating Systems Support for Digital Audio and Video*. 2009, pp. 37–42.

[62] H. Shojania, B. Li, and X. Wang. “Nuclei: GPU-Accelerated Many-Core Network Coding”. In: *Proceedings of the 28th IEEE Int. Conference on Computer Communications*. 2009, pp. 459–467.

[63] J. K. Sundararajan, D. Shah, M. Médard, M. Mitzenmacher, and J. Barros. “Network Coding Meets TCP”. In: *Proceedings of the 28th IEEE Int. Conference on Computer Communications*. 2009, pp. 280–288.

[64] M. Alizadeh, A. Greenberg, D. Maltz A, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. “Data Center TCP (DCTCP)”. In: *Proceedings of the 24th ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. 2010, pp. 63–74.

[65] J. Baliga, R. W. A. Ayre, K. Hinton, and R. S. Tucker. “Green Cloud Computing: Balancing Energy in Processing, Storage, and Transport”. In: *Proceedings of the IEEE* 99.1 (2010), pp. 149–167.

[66] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. “Netalyzr: Illuminating the Edge Network”. In: *Proceedings of the 10th ACM Internet Measurement Conference*. 2010, pp. 246–259.

[67] D. Mills, J. Martin, J. Burbank, and W. Kasch. *Network Time Protocol Version 4: Protocol and Algorithms Specification*. RFC 5905. IETF, 2010.

[68] B. H. Oh, J. Han, and J. Lee. “Timer and Sequence Based Packet Loss Detection Scheme for Efficient Selective Retransmission in DCCP”. In: *Proceedings of the 5th Int. Conference on Future Generation Communication and Networking*. Springer. 2010, pp. 112–120.

[69] J. D. Opdyke and John Douglas. “A Unified Approach to Algorithms Generating Unrestricted and Restricted Integer Compositions and Integer Partitions”. In: *Journal of Mathematical Modelling and Algorithms* 9.1 (2010), pp. 53–97.

[70] Y. Polyanskiy, H. V. Poor, and S. Verdú. “Channel Coding Rate in the Finite Blocklength Regime”. In: *IEEE Transactions on Information Theory* 56.5 (2010), pp. 2307–2359.

[71] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic. “Cyber-Physical Systems: The Next Computing Revolution”. In: *Proceedings of the 47th Design Automation Conference*. 2010, pp. 731–736.

[72] H. Shojania and B. Li. “Tenor: Making Coding Practical from Servers to Smartphones”. In: *Proceedings of the 18th ACM Int. Conference on Multimedia*. 2010, pp. 45–54.

[73] L. Torrey and J. Shavlik. “Transfer Learning”. In: *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*. IGI Global, 2010, pp. 242–264.

[74] J. Gettys. “Bufferbloat: Dark Buffers in the Internet”. In: *IEEE Internet Computing* 15.3 (2011), pp. 96–96.

[75] J. Gettys and K. Nichols. “Bufferbloat: Dark Buffers in the Internet: Networks without Effective AQM May Again Be Vulnerable to Congestion Collapse.” In: *ACM Queue Magazine* 9.11 (2011), pp. 40–54.

[76] S. S. Haykin and M. Moher. *Modern Wireless Communications*. Pearson Education India, 2011.

[77] *ITU-T Y.1541: Network Performance Objectives for IP-Based Services*. Tech. rep. ITU, 2011.

[78] S. Kaul, M. Gruteser, V. Rai, and J. Kenney. “Minimizing Age of Information in Vehicular Networks”. In: *Proceedings of the 8th IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*. 2011, pp. 350–358.

[79] V. Paxson, M. Allman, J. Chu, and M. Sargent. *Computing TCP’s Retransmission Timer*. RFC 6298. IETF, 2011.

[80] S. M. Rumble, D. Ongaro, R. Stutsman, M. Rosenblum, and J. K. Ousterhout. “It’s Time for Low Latency”. In: *Proceedings of the 13th Workshop on Hot Topics in Operating Systems*. 2011, pp. 1–11.

[81] T. Stockhammer. “Dynamic Adaptive Streaming over HTTP– Standards and Design Principles”. In: *Proceedings of the 2nd ACM Conference on Multimedia Systems*. 2011, pp. 133–144.

[82] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron. “Better Never than Late: Meeting Deadlines in Datacenter Networks”. In: *ACM SIGCOMM Computer Communication Review* 41.4 (2011), pp. 50–61.

[83] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda. "Less is More: Trading a Little Bandwidth for Ultra-Low Latency in the Data Center". In: *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation*. 2012, pp. 253–266.

[84] M. Allman. "Comments on Bufferbloat". In: *ACM SIGCOMM Computer Communication Review* 43.1 (2012), pp. 30–37.

[85] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. "Fog Computing and Its Role in the Internet of Things". In: *Proceedings of the 1st MCC Workshop on Mobile Cloud Computing*. 2012, pp. 13–16.

[86] M. Gorius. "Adaptive Delay-Constrained Internet Media Transport". PhD thesis. Saarland Informatics Campus, 2012.

[87] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida. *The NewReno Modification to TCP's Fast Recovery Algorithm*. RFC 6582. IETF, 2012.

[88] T. Hönig, C. Eibel, R. Kapitza, and W. Schröder-Preikschat. "SEEP: Exploiting Symbolic Execution for Energy-Aware Programming". In: *ACM SIGOPS Operating Systems Review* 45.3 (2012), pp. 58–62.

[89] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari. "Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard". In: *Proceedings of the 12th ACM Internet Measurement Conference*. 2012, pp. 225–238.

[90] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 26th Conference on Neural Information Processing Systems* 25 (2012).

[91] B. Vamanan, J. Hasan, and T. N. Vijaykumar. "Deadline-Aware Datacenter TCP (D2TCP)". In: *Proceedings of the 26th ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* 42.4 (2012), pp. 115–126.

[92] J. Cloud, F. du Pin Calmon, W. Zeng, G. Pau, L. M. Zeger, and M. Medard. "Multi-Path TCP with Network Coding for Mobile Devices in Heterogeneous Networks". In: *Proceedings of the 78th IEEE Vehicular Technology Conference*. 2013, pp. 1–5.

[93] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan. "Reducing Web Latency: The Virtue of Gentle Aggression". In: *Proceedings of the 27th ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. 2013, pp. 159–170.

[94] S. Lederer, C. Mueller, C. Timmerer, C. Concolato, J. Le Feuvre, and K. Fliegel. "Distributed DASH Dataset". In: *Proceedings of the 4th ACM Multimedia Systems Conference*. 2013, pp. 131–135.

[95] J. Qureshi, C. Heng Foh, and J. Cai. "Primer and Recent Developments on Fountain Codes". In: *Recent Advances in Communications and Networking Technology* 2.1 (2013), pp. 2–11.

[96] J. Ba and R. Caruana. "Do Deep Nets Really Need to Be Deep?" In: *Proceedings of the 28th Conference on Neural Information Processing Systems* 27 (2014), pp. 1–9.

[97] T. Hönig, H. Janker, C. Eibel, O. Mihelic, and R. Kapitza. "Proactive Energy-Aware Programming with PEEK". In: *Proceedings of the 2nd Conference on Timely Results in Operating Systems*. 2014, pp. 1–14.

[98] M. González De Molina and V. M. Toledo. *The Social Metabolism: A Socio-Ecological Theory of Historical Change*. Vol. 3. Springer, 2014.

[99] A. Saifullah, C. Wu, P. B. Tiwari, Y. Xu, Y. Fu, C. Lu, and Yixin Y. Chen. "Near Optimal Rate Selection for Wireless Control Systems". In: *ACM Transactions on Embedded Computing Systems* 13.4s (2014), pp. 1–25.

[100] A. Singla, B. Chandrasekaran, P. B. Godfrey, and Bruce B. Maggs. "The Internet at the Speed of Light". In: *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*. 2014, pp. 1–7.

[101] *The Tactile Internet - ITU-T Technology Watch Report*. Tech. rep. ITU, 2014.

[102] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and Michael M. Schapira. "PCC: Re-Architecting Congestion Control for Consistent High Performance". In: *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation*. 2015, pp. 395–408.

[103] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: *Proceedings of the 3rd Int. Conference on Learning Representations*. 2015, pp. 1–15.

[104] S. Kounev, X. Zhu, J. O. Kephart, and M. Kwiatkowska. *Model-Driven Algorithms and Architectures for Self-Aware Computing Systems*. Dagstuhl Seminar 15041. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2015.

[105] S. Li, L. D. Xu, and S. Zhao. "The Internet of Things: A Survey". In: *Information Systems Frontiers* 17.2 (2015), pp. 243–259.

[106] K. Liu and J. Y. B. Lee. "On Improving TCP Performance over Mobile Data Networks". In: *IEEE Transactions on Mobile Computing* 15.10 (2015), pp. 2522–2536.

[107] Y. Liu, S. Dey, F. Ulupinar, M. Luby, and Y. Mao. "Deriving and Validating User Experience Model for DASH Video Streaming". In: *IEEE Transactions on Broadcasting* 61.4 (2015), pp. 651–665.

[108] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats. "TIMELY: RTT-Based Congestion Control for the Datacenter". In: *ACM SIGCOMM Computer Communication Review* 45.4 (2015), pp. 537–550.

[109] M. Rajiullah, P. Hurtig, A. Brunstrom, A. Petlund, and M. Welzl. "An Evaluation of Tail Loss Recovery Mechanisms for TCP". In: *ACM SIGCOMM Computer Communication Review* 45.1 (2015), pp. 5–11.

[110] A. Torres, C. T. Calafate, J. C. Cano, P. Manzoni, and Y. Ji. "Evaluation of Flooding Schemes for Real-Time Video Transmission in VANETs". In: *Ad Hoc Networks Journal* 24 (2015), pp. 3–20.

[111] A. Belay, G. Prekas, M. Primorac, A. Klimovic, S. Grossman, C. Kozyrakis, and E. Bugnion. "The IX Operating System: Combining Low Latency, High Throughput, and Efficiency in a Protected Dataplane". In: *ACM Transactions on Computer Systems* 34.4 (2016), pp. 1–39.

[112] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and Van V. Jacobson. "BBR: Congestion-Based Congestion Control: Measuring Bottleneck Bandwidth and Round-Trip Propagation Time". In: *ACM Queue Magazine* 14.5 (2016), pp. 20–53.

[113] M. Claeys, N. Bouten, D. De Vleeschauwer, K. De Schepper, W. Van Leekwijck, S. Latré, and F. De Turck. "Deadline-Aware TCP Congestion Control for Video Streaming Services". In: *Proceedings of the 12th IEEE Int. Conference on Network and Service Management*. 2016, pp. 100–108.

[114] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT press, 2016.

[115] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. "Binarized Neural Networks". In: *Proceedings of the 30th Conference on Neural Information Processing Systems* 29 (2016), pp. 1–9.

[116] E. A. Lee and S. A. Seshia. *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*. MIT press, 2016.

[117] D. Li, X. Chen, M. Becchi, and Z. Zong. "Evaluating the Energy Efficiency of Deep Convolutional Neural Networks on CPUs and GPUs". In: *Proceedings of the 3rd IEEE Int. Conferences on Big Data and Cloud Computing, Social Computing and Networking, Sustainable Computing and Communications*. 2016, pp. 477–484.

[118] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. "Learning Structured Sparsity in Deep Neural Networks". In: *Proceedings of the 30th Conference on Neural Information Processing Systems* 29 (2016), pp. 1–9.

[119] T. Azzino, M. Drago, M. Polese, A. Zanella, and M. Zorzi. "X-TCP: A Cross Layer Approach for TCP Uplink Flows in mmWave Networks". In: *Proceedings of the 16th IEEE Annual Mediterranean Ad Hoc Networking Workshop*. 2017, pp. 1–6.

[120] A. Bhartia, B. Chen, F. Wang, D. Pallas, R. Musaloiu-E, T. T.-T. Lai, and H. Ma. "Measurement-Based, Practical Techniques to Improve 802.11ac Performance". In: *Proceedings of the 17th Internet Measurement Conference*. 2017, pp. 205–219.

[121] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo. “Congestion Control for Web Real-Time Communication”. In: *IEEE/ACM Transactions on Networking* 25.5 (2017), pp. 2629–2642.

[122] R. M. Jungers, A. Kundu, and W. P. M. H. Heemels. “Observability and Controllability Analysis of Linear Systems Subject to Data Losses”. In: *IEEE Transactions on Automatic Control* 63.10 (2017), pp. 3361–3376.

[123] M. Karzand, D. J. Leith, J. Cloud, and M. Medard. “Design of FEC for Low Delay in 5G”. In: *IEEE Journal on Selected Areas in Communications* 35.8 (2017), pp. 1783–1793.

[124] S. Linsenmayer and F. Allgower. “Stabilization of Networked Control Systems with Weakly Hard Real-Time Dropout Description”. In: *Proceedings of the 56th IEEE Conference on Decision and Control*. 2017, pp. 4765–4770.

[125] S. Mangiante, G. Klas, A. Navon, Z. GuanHua, J. Ran, and M. D. Silva. “VR is on the Edge: How to Deliver 360 Videos in Mobile Networks”. In: *Proceedings of the 1st Workshop on Virtual Reality and Augmented Reality Network*. 2017, pp. 30–35.

[126] H. Mao, R. Netravali, and M. Alizadeh. “Neural Adaptive Video Streaming with Pensieve”. In: *Proceedings of the 31st ACM SIGCOMM Conference*. 2017, pp. 197–210.

[127] M. Polese, M. Mezzavilla, M. Zhang, J. Zhu, S. Rangan, S. Panwar, and M. Zorzi. “milliProxy: A TCP Proxy Architecture for 5G mmWave Cellular Systems”. In: *Proceedings of the 51st IEEE Asilomar Conference on Signals, Systems, and Computers*. 2017, pp. 951–957.

[128] J. A. Preiss, W. Honig, G. S. Sukhatme, and N. Ayanian. “Crazyswarm: A Large Nano-Quadcopter Swarm”. In: *Proceedings of the 33rd IEEE Int. Conference on Robotics and Automation*. 2017, pp. 3299–3304.

[129] V. Roca, B. Teibi, C. Burdinat, T. Tran, and C. Thienot. “Less Latency and Better Protection with AL-FEC Sliding Window Codes: A Robust Multimedia CBR Broadcast Case Study”. In: *Proceedings of the 13th IEEE Int. Conference on Wireless and Mobile Computing, Networking and Communications*. 2017, pp. 1–8.

[130] A. Schmidt and T. Herfet. “Transparent Transmission Segmentation in Software-Defined Networks”. In: *Proceedings of the 3rd IEEE Conference on Network Softwarization*. 2017, pp. 1–5.

[131] R. K. Sheshadri and D. Koutsonikolas. “On Packet Loss Rates in Modern 802.11 Networks”. In: *Proceedings of the 36th IEEE IEEE Int. Conference on Computer Communications*. 2017, pp. 1–9.

[132] S. Wunderlich, J. A. Cabrera, F. H. P. Fitzek, and M. Reisslein. “Network Coding in Heterogeneous Multicore IoT Nodes with DAG Scheduling of Parallel Matrix Block Operations”. In: *IEEE Internet of Things Journal* 4.4 (2017), pp. 917–933.

[133] M. Zhu and S. Gupta. "To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression". In: *arXiv* (2017), pp. 1–11.

[134] V. Arun and H. Balakrishnan. "Copa: Practical Delay-Based Congestion Control for the Internet". In: *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation*. 2018, pp. 329–342.

[135] S. Bateni, H. Zhou, Y. Zhu, and C. Liu. "Predjoule: A Timing-Predictable Energy Optimization Framework for Deep Neural Networks". In: *Proceedings of the 39th IEEE Real-Time Systems Symposium*. 2018, pp. 107–118.

[136] H. S. Chwa, K. G. Shin, and J. Lee. "Closing the gap between stability and schedulability: A new task model for cyber-physical systems". In: *Proceedings of the 24th IEEE Real-Time and Embedded Technology and Applications Symposium*. 2018, pp. 327–337.

[137] D. D. Clark. *Designing an Internet*. MIT Press, 2018.

[138] S. Ferlin, S. Kucera, H. Claussen, and Ö. Alay. "MPTCP Meets FEC: Supporting Latency-Sensitive Applications over Heterogeneous Networks". In: *IEEE/ACM Transactions on Networking* 26.5 (2018), pp. 2005–2018.

[139] N. Finn. "Introduction to Time-Sensitive Networking". In: *IEEE Communications Standards Magazine* 2.2 (2018), pp. 22–28.

[140] T. Hönig, C. Eibel, A. Wagenhäuser, M. Wagner, and W. Schröder-Preikschat. "How to Make Profit: Exploiting Fluctuating Electricity Prices with Albatross, a Runtime System for Heterogeneous HPC Clusters". In: *Proceedings of the 8th ACM Int. Workshop on Runtime and Operating Systems for Supercomputers*. 2018, pp. 1–9.

[141] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and S. Kalenichenko. "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-only Inference". In: *Proceedings of the 27th IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2704–2713.

[142] A. Narayan, F. Cangialosi, D. Raghavan, P. Goyal, S. Narayana, R. Mittal, M. Alizadeh, and H. Balakrishnan. "Restructuring Endpoint Congestion Control". In: *Proceedings of the 32nd ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. 2018, pp. 30–43.

[143] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury. "Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research". In: *IEEE Communications Surveys & Tutorials* 21.1 (2018), pp. 88–145.

[144] M. Palmer, T. Krüger, B. Chandrasekaran, and A. Feldmann. "The QUIC Fix For Optimal Video Streaming". In: *Proceedings of the 1st ACM Workshop on the Evolution, Performance, and Interoperability of QUIC*. 2018, pp. 43–49.

[145] D. Raca, J. J. Quinlan, A. H. Zahran, and C. J. Sreenan. "Beyond Throughput: A 4G LTE Dataset with Channel and Context Metrics". In: *Proceedings of the 9th ACM Multimedia Systems Conference*. 2018, pp. 460–465.

[146] S. Xiao, D. He, and Z. Gong. "Deep-Q: Traffic-Driven QoS Inference Using Deep Generative Network". In: *Proceedings of the 1st ACM Workshop on Network Meets AI & ML*. 2018, pp. 67–73.

[147] F. Y. Yan, J. Ma, G. D. Hill, D. Raghavan, R. S. Wahby, P. Levis, and K. Winstein. "Pantheon: The Training Ground for Internet Congestion-Control Research". In: *Proceedings of the 29th USENIX Annual Technical Conference*. 2018, pp. 731–743.

[148] Q. De Coninck, F. Michel, M. Piraux, F. Rochet, T. Given-Wilson, A. Legay, O. Pereira, and O. Bonaventure. "Pluginizing QUIC". In: *Proceedings of the 33rd ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. 2019, pp. 59–74.

[149] N. Finn, P. Thubert, B. Varga, and J. Farkas. *Deterministic Networking Architecture*. RFC 8655. IETF, 2019.

[150] S. L. Fong, A. Khisti, B. Li, W.-T. Tan, X. Zhu, and J. Apostolopoulos. "Optimal Streaming Codes for Channels with Burst and Arbitrary Erasures". In: *IEEE Transactions on Information Theory* 65.7 (2019), pp. 4274–4292.

[151] P. Garrido, I. Sanchez, S. Ferlin, R. Aguero, and O. Alay. "rQUIC: Integrating FEC with QUIC for Robust Wireless Communications". In: *Proceedings of the 34th IEEE Global Communications Conference*. 2019, pp. 1–7.

[152] D. Ginthör, J. von Hoyningen-Huene, R. Guillaume, and H. Schotten. "Analysis of Multi-User Scheduling in a TSN-Enabled 5G System for Industrial Applications". In: *Proceedings of the 2nd IEEE Int. Conference on Industrial Internet*. 2019, pp. 190–199.

[153] T. Hönig, B. Herzog, and W. Schröder-Preikschat. "Energy-Demand Estimation of Embedded Devices Using Deep Artificial Neural Networks". In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. 2019, pp. 617–624.

[154] J. Hu, J. Huang, W. Lv, Y. Zhou, J. Wang, and T. He. "CAPS: Coding-Based Adaptive Packet Spraying to Reduce Flow Completion Time in Data Center". In: *IEEE/ACM Transactions on Networking* 27.6 (2019), pp. 2338–2353.

[155] T. Huang, C. Zhou, R.-X. Zhang, C. Wu, X. Yao, and L. Sun. "Comyco: Quality-Aware Adaptive Video Streaming via Imitation Learning". In: *Proceedings of the 27th ACM Int. Conference on Multimedia*. 2019, pp. 429–437.

[156] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. IEEE 1588-2019. IEEE, 2019.

[157] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar. "A Deep Reinforcement Learning Perspective on Internet Congestion Control". In: *Proceedings of the 36th Int. Conference on Machine Learning*. 2019, pp. 3050–3059.

[158] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, and M. Yu. "HPCC: High Precision Congestion Control". In: *Proceedings of the 33rd ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. 2019, pp. 44–58.

[159] K. N. McGuire, C. De Wagter, K. Tuyls, H. J. Kappen, and G. C. H. E. de Croon. "Minimal Navigation Solution for a Swarm of Tiny Flying Robots to Explore an Unknown Environment". In: *Science Robotics Journal* 4.35 (2019), pp. 1–14.

[160] F. Michel, Q. De Coninck, and O. Bonaventure. "QUIC-FEC: Bringing the Benefits of Forward Erasure Correction to QUIC". In: *Proceedings of the 7th IEEE IFIP Networking Conference*. 2019, pp. 1–9.

[161] A. Ousterhout, A. Belay, and I. Zhang. "Just in Time Delivery: Leveraging Operating Systems Knowledge for Better Datacenter Congestion Control". In: *Proceedings of the 11th USENIX Workshop on Hot Topics in Cloud Computing*. 2019, pp. 1–9.

[162] A. Ousterhout, J. Fried, J. Behrens, A. Belay, and Hari H. Balakrishnan. "Shenango: Achieving High CPU Efficiency for Latency-Sensitive Datacenter Workloads". In: *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation*. 2019, pp. 361–378.

[163] A. Schmidt. "Cross-Layer Latency-Aware and -Predictable Data Communication". PhD thesis. Saarland Informatics Campus, 2019.

[164] L. N. Smith and N. Topin. "Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates". In: *Proceedings of the 31st SPIE Defense & Commercial Sensing Conference*. 2019, pp. 369–386.

[165] E. Strubell, A. Ganesh, and A. McCallum. "Energy and Policy Considerations for Deep Learning in NLP". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2019, pp. 3645–3650.

[166] J. Xia, G. Zeng, J. Zhang, W. Wang, W. Bai, J. Jiang, and K. Chen. "Rethinking Transport Layer Design for Distributed Machine Learning". In: *Proceedings of the 3rd ACM Asia-Pacific Workshop on Networking*. 2019, pp. 22–28.

[167] M. Xu, J. Liu, Y. Liu, F. X. Lin, Y. Liu, and X. Liu. "A First Look at Deep Learning Apps on Smartphones". In: *Proceedings of the 1st ACM Web Conference*. 2019, pp. 2125–2136.

[168] A. Zappone, M. Di Renzo, M. Debbah, T. Lam Tu, and X. Qian. "Model-Aided Wireless Artificial Intelligence: Embedding Expert Knowledge in Deep Neural Networks for Wireless System Optimization". In: *IEEE Vehicular Technology Magazine* 14.3 (2019), pp. 60–69.

[169] D. Zarchy, R. Mittal, M. Schapira, and S. Shenker. "Axiomatizing Congestion Control". In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 3.2 (2019), pp. 1–33.

[170] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. "Language Models Are Few-Shot Learners". In: *Proceedings of the 34th Conference on Neural Information Processing Systems* 33 (2020), pp. 1877–1901.

[171] S. Cheng, H. Hu, X. Zhang, and Z. Guo. "DeepRS: Deep-Learning Based Network-Adaptive FEC for Real-Time Video Communications". In: *Proceedings of the 32nd IEEE Int. Symposium on Circuits and Systems*. 2020, pp. 1–5.

[172] A. Cohen, D. Malak, V. B. Bracha, and M. Médard. "Adaptive Causal Network Coding with Feedback". In: *IEEE Transactions on Communications* 68.7 (2020), pp. 4325–4341.

[173] E. Dahlman, S. Parkvall, and Johan J. Skold. *5G NR: The Next Generation Wireless Access Technology*. Academic Press, 2020.

[174] S. Emara, B. Li, and Y. Chen. "Eagle: Refining Congestion Control by Learning from the Experts". In: *Proceedings of the 39th IEEE Int. Conference on Computer Communications*. 2020, pp. 676–685.

[175] P. Goyal, A. Agarwal, R. Netravali, M. Alizadeh, and H. Balakrishnan. "ABC: A Simple Explicit Congestion Controller for Wireless Networks". In: *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation*. 2020, pp. 353–372.

[176] S. Kumar, M. P. Andersen, H.-S. Kim, and D. E. Culler. "Performant TCP for Low-Power Wireless Networks". In: *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation*. 2020, pp. 911–932.

[177] M. Maggio, A. Hamann, E. Mayer-John, and D. Ziegenbein. "Control-System Stability under Consecutive Deadline Misses Constraints". In: *Proceedings of the 32nd Euromicro Conference on Real-Time Systems*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2020, 21:1–21:24.

[178] A. Narayanan, E. Ramadan, J. Carpenter, Q. Liu, Y. Liu, F. Qian, and Z.-L. Zhang. "A First Look at Commercial 5G Performance on Smartphones". In: *Proceedings of the 2nd ACM Web Conference*. 2020, pp. 894–905.

[179] S. Reif, B. Herzog, J. Hemp, T. Hönig, and W. Schröder-Preikschat. "Precious: Resource-Demand Estimation for Embedded Neural Network Accelerators". In: *Proceedings of the 1st Int. Workshop on Benchmarking Machine Learning Workloads on Emerging Hardware*. 2020, pp. 1–9.

[180] S. Reif and Wolfgang W: Schröder-Preikschat. "Precisely Timed Task Execution". In: *Proceedings of the 23rd IEEE Int. Symposium on Real-Time Distributed Computing*. 2020, pp. 10–19.

[181] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni. “Green AI”. In: *Communications of the ACM* 63.12 (2020), pp. 54–63.

[182] M.P. Sharabayko, M.A. Sharabayko, J. Dube, JS. Kim, and JW. Kim. *The SRT Protocol*. Internet Draft. IETF, 2020.

[183] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. “BOLA: Near-Optimal Bitrate Adaptation for Online Videos”. In: *IEEE/ACM Transactions on Networking* 28.4 (2020), pp. 1698–1711.

[184] D. Xu, A. Zhou, X. Zhang, G. Wang, X. Liu, C. An, Y. Shi, L. Liu, and H. Ma. “Understanding Operational 5G: A First Measurement Study on Its Coverage, Performance and Energy Consumption”. In: *Proceedings of the 34th ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. 2020, pp. 479–494.

[185] T. Zeng, O. Semiari, M. Mozaffari, M. Chen, W. Saad, and M. Bennis. “Federated Learning in the Sky: Joint Power Allocation and Scheduling with UAV Swarms”. In: *Proceedings of the 30th IEEE Int. Conference on Communications*. 2020, pp. 1–6.

[186] A. Agarwal, J. Sun, S. Noghabi, S. Iyengar, A. Badam, R. Chandra, S. Seshan, and S. Kalyanaraman. “Redesigning Data Centers for Renewable Energy”. In: *Proceedings of the 20th ACM Workshop on Hot Topics in Networks*. 2021, pp. 45–52.

[187] Y. Cheng, N. Cardwell, N. Dukkipati, and P. Jha. *The RACK-TLP Loss Detection Algorithm for TCP*. RFC 8985. IETF, 2021.

[188] S. Emara, S. L. Fong, B. Li, A. Khisti, W.-T. Tan, X. Zhu, and J. Apostolopoulos. “Low-Latency Network-Adaptive Error Control for Interactive Streaming”. In: *IEEE Transactions on Multimedia* 24 (2021), pp. 1691–1706.

[189] M.-A. Gütschow. *Low-Level Design of Energy-Efficient HARQ at the Transport Layer*. Bachelor thesis. 2021.

[190] B. Herzog, F. Hügel, S. Reif, T. Hönig, and W. Schröder-Preikschat. “Automated Selection of Energy-Efficient Operating System Configurations”. In: *Proceedings of the 12th ACM Int. Conference on Future Energy Systems*. 2021, pp. 309–315.

[191] H. Hu, S. Cheng, X. Zhang, and Z. Guo. “LightFEC: Network Adaptive FEC with a Lightweight Deep-Learning Approach”. In: *Proceedings of the 29th ACM Int. Conference on Multimedia*. 2021, pp. 3592–3600.

[192] L. Zhang Lyna, S. Han, J. Wei, N. Zheng, T. Cao, Y. Yang, and Y. Liu. “nn-Meter: Towards Accurate Latency Prediction of Deep-Learning Model Inference on Diverse Edge Devices”. In: *Proceedings of the 19th Int. Conference on Mobile Systems, Applications, and Services*. 2021, pp. 81–93.

[193] M. Palmer, M. Appel, K. Spiteri, B. Chandrasekaran, A. Feldmann, and R. K. Sitaraman. "VOXEL: Cross-Layer Optimization for Video Streaming with Imperfect Transmission". In: *Proceedings of the 17th Int. Conference on Emerging Networking Experiments and Technologies*. 2021, pp. 359–374.

[194] J. Pineau, P. Vincent-Lamarre, K. Sinha, V. Larivière, A. Beygelzimer, F. d'Alché Buc, E. Fox, and H. Larochelle. "Improving Reproducibility in Machine Learning Research (A Report from the NeueIPS 2019 Reproducibility Program)". In: *Journal of Machine Learning Research* 22.1 (2021), pp. 7459–7478.

[195] A. Radovanovic, R. Koningstein, I. Schneider, B. Chen, A. Duarte, B. Roy, D. Xiao, M. Haridasan, P. Hung, Nick N. Care, S. Talukdar, E. Mullen, K. Smith, M. Cottman, and W. Cirne. "Carbon-Aware Computing for Datacenters". In: *IEEE Transactions on Power Systems* 38.2 (2021), pp. 1270–1280.

[196] N. Vreman, A. Cervin, and M. Maggio. "Stability and Performance Analysis of Control Systems Subject to Bursts of Deadline Misses". In: *Proceedings of the 33rd Euromicro Conference on Real-Time Systems*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2021, 15:1–15:23.

[197] L. Zhang, Y. Cui, J. Pan, and Y. Jiang. "Deadline-Aware Transmission Control for Real-Time Video Streaming". In: *Proceedings of the 29th IEEE Int. Conference on Network Protocols*. 2021, pp. 1–6.

[198] M. Zverev, P. Garrido, F. Fernández, J. Bilbao, Ö. Alay, S. Ferlin, A. Brunstrom, and R. Agüero. "Robust QUIC: Integrating Practical Coding in a Low Latency Transport Protocol". In: *IEEE Access* 9 (2021), pp. 138225–138244.

[199] T. Anderson, A. Belay, M. Chowdhury, A. Cidon, and I. Zhang. "Treehouse: A Case For Carbon-Aware Datacenter Software". In: *ACM SIGENERGY Energy Informatics Review* 3.3 (2022), pp. 64–70.

[200] M. Bishop. *HTTP/3*. RFC 9114. IETF, 2022.

[201] N. Cardwell, Y. Cheng, S. Hassas Yeganeh, I. Swett, and V. Jacobson. *BBR Congestion Control*. Internet Draft. IETF, 2022.

[202] K. Chen, H. Wang, S. Fang, X. Li, M. Ye, and H. J. Chao. "RL-AFEC: Adaptive Forward Error Correction for Real-Time Video Communication Based on Reinforcement Learning". In: *Proceedings of the 13th ACM Multimedia Systems Conference*. 2022, pp. 96–108.

[203] Y. Cheng, N. Cardwell, S. H. Yeganeh, and V. Jacobson. *Delivery Rate Estimation*. Internet Draft. IETF, 2022.

[204] W. Dong, J. Lv, G. Chen, Y. Wang, H. Li, Y. Gao, and D. Bharadia. "TinyNet: A Lightweight, Modular, and Unified Network Architecture for the Internet of Things". In: *Proceedings of the 20th Int. Conference on Mobile Systems, Applications and Services*. 2022, pp. 248–260.

[205] W. Eddy. *Transmission Control Protocol (TCP)*. RFC 9293. IETF, 2022.

[206] S. Emara, F. Wang, I. Kaplan, and B. Li. "Ivory: Learning Network Adaptive Streaming Codes". In: *Proceedings of the 30th IEEE/ACM Int. Symposium on Quality of Service*. 2022, pp. 1–10.

[207] A. Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, Inc., 2022.

[208] B. Herzog, S. Reif, J. Hemp, T. Hönig, and W. Schröder-Preikschat. "Resource-Demand Estimation for Edge Tensor Processing Units". In: *ACM Transactions on Embedded Computing Systems* 21.5 (2022), pp. 1–24.

[209] B. Josephrexon Jeniefer and M. Maggio. "Experimenting with Networked Control Software Subject to Faults". In: *Proceedings of the 61st IEEE Conference on Decision and Control*. 2022, pp. 1547–1552.

[210] Y. Ma, H. Tian, X. Liao, J. Zhang, W. Wang, K. Chen, and X. Jin. "Multi-Objective Congestion Control". In: *Proceedings of the 17th European Conference on Computer Systems*. 2022, pp. 218–235.

[211] F. Michel, A. Cohen, D. Malak, Q. De Coninck, M. Médard, and O. Bonaventure. "FlEC: Enhancing QUIC with Application-Tailored Reliability Mechanisms". In: *IEEE/ACM Transactions on Networking* 2 (2022), pp. 606–619.

[212] T. Pauly, E. Kinnear, and D. Schinazi. *An Unreliable Datagram Extension to QUIC*. RFC 9221. IETF, 2022.

[213] G. Siracusano, S. Galea, D. Sanvito, M. Malekzadeh, G. Antichi, P. Costa, H. Haddadi, and R. Bifulco. "Re-Architecting Traffic Analysis with Neural Network Interface Cards". In: *Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation*. 2022, pp. 513–533.

[214] H. Tian, X. Liao, C. Zeng, J. Zhang, and K. Chen. "Spine: An Efficient DRL-Based Congestion Control with Ultra-Low Overhead". In: *Proceedings of the 18th Int. Conference on Emerging Networking Experiments and Technologies*. 2022, pp. 261–275.

[215] J. Zhang, H. Shi, Y. Cui, F. Qian, W. Wang, K. Zheng, and J. Wu. "To Punctuality and Beyond: Meeting Application Deadlines with DTP". In: *Proceedings of the 30th IEEE Int. Conference on Network Protocols*. 2022, pp. 1–11.

[216] N. Zilberman, E. M. Schooler, U. Cummings, R. Manohar, D. Nafus, R. Soulé, and R. Taylor. "Toward Carbon-Aware Networking". In: *ACM SIGENERGY Energy Informatics Review* 3.3 (2022), pp. 15–20.

[217] B. Arratia, J. Prades, S. Peña-Haro, J. M. Cecilia, and P. Manzoni. "BODOQUE: An Energy-Efficient Flow Monitoring System for Ephemeral Streams". In: *Proceedings of the 24th ACM Int. Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*. 2023, pp. 358–363.

[218] Z. Chen, L. Shi, X. Liu, X. Ai, S. Liu, and Y. Xu. "Boosting Distributed Machine Learning Training Through Loss-Tolerant Transmission Protocol". In: *Proceedings of the 31st IEEE/ACM Int. Symposium on Quality of Service*. 2023, pp. 1–4.

[219] IPCC. *Climate Change 2023: Synthesis Report. Contribution of Working Groups I, II and III to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change [Core Writing Team, H. Lee and J. Romero (eds.)]* Tech. rep. IPCC, Geneva, Switzerland, 2023, pp. 35–115.

[220] R. Jacob and L. Vanbever. "The Internet of Tomorrow Must Sleep More and Grow Old". In: *ACM SIGENERGY Energy Informatics Review* 3 (2023), pp. 27–32.

[221] N. Maslej, L. Fattorini, E. Brynjolfsson, J. Etchemendy, K. Ligett, T. Lyons, J. Manyika, H. Ngo, J. C. Niebles, V. Parli, Y. Shoham, R. Wald, J. Clark, and R. Perrault. *The AI Index 2023 Annual Report*. Tech. rep. Institute for Human-Centered AI, Stanford University, 2023.

[222] J. Park, K. Han, and B. Lee. "Green Cloud? An Empirical Analysis of Cloud Computing and Energy Efficiency". In: *Management Science* 69.3 (2023), pp. 1639–1664.

[223] T. Pauly, B. Trammell, A. Brunstrom, G. Fairhurst, and C. Perkins. *An Architecture for Transport Services*. Internet Draft. IETF, 2023.

[224] T. Rheinfels, M. Gaukler, and P. Ulbrich. "A New Perspective on Criticality: Efficient State Abstraction and Run-Time Monitoring of Mixed-Criticality Real-Time Control Systems". In: *Proceedings of the 35th Euromicro Conference on Real-Time Systems*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik. 2023.

[225] S. Senk, H. K. Nazari, H.-H. Liu, G. T. Nguyen, and F. H. P. Fitzek. "Open-Source Testbeds for Integrating Time-Sensitive Networking with 5G and beyond". In: *Proceedings of the 20th IEEE Consumer Communications & Networking Conference*. 2023, pp. 1–7.

[226] V. Tripathi, I. Kadota, E. Tal, M. S. Rahman, A. Warren, S. Karaman, and E. Modiano. "WiSwarm: Age-of-Information-Based Wireless Networking for Collaborative Teams of UAVs". In: *Proceedings of the 42nd IEEE Int. Conference on Computer Communications*. 2023.

[227] Y. Zhao and T. Guo. "Carbon-Efficient Neural Architecture Search". In: *Proceedings of the 2nd Workshop on Sustainable Computer Systems Design and Implementation*. 2023.